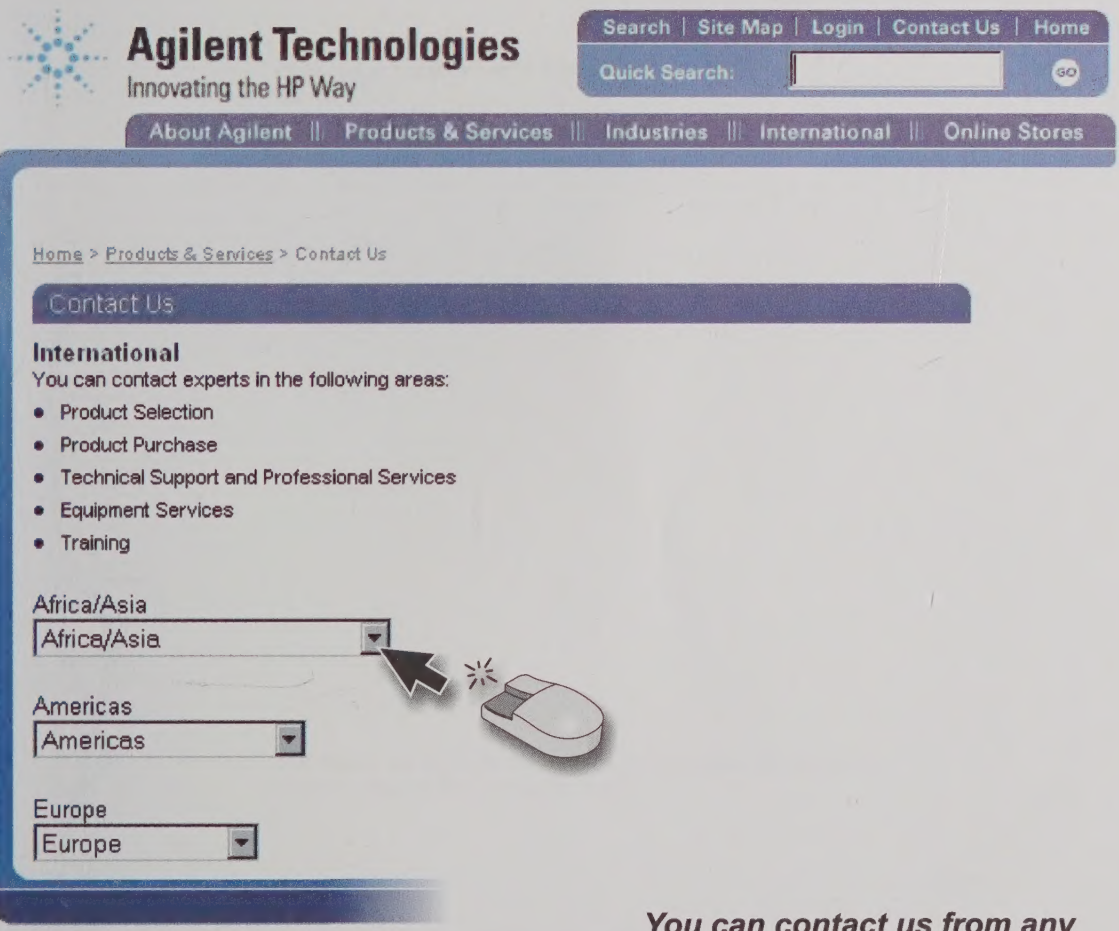
**TBM Test**
2613 Skinner Dr.
Flower Mound, TX
75028
972-691-9708

Agilent Technologies

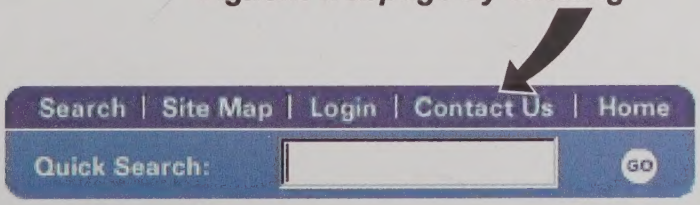
Programmer's Reference

# Infiniium Oscilloscopes

# Need Help? www.agilent.com/find/need-help




***You can contact us from any Agilent webpage by clicking***




16700-92011

Digitized by the Internet Archive
in 2025

https://archive.org/details/manualsplus_09041

# Programmer's Reference

Publication Number 54810-97076
March 2002

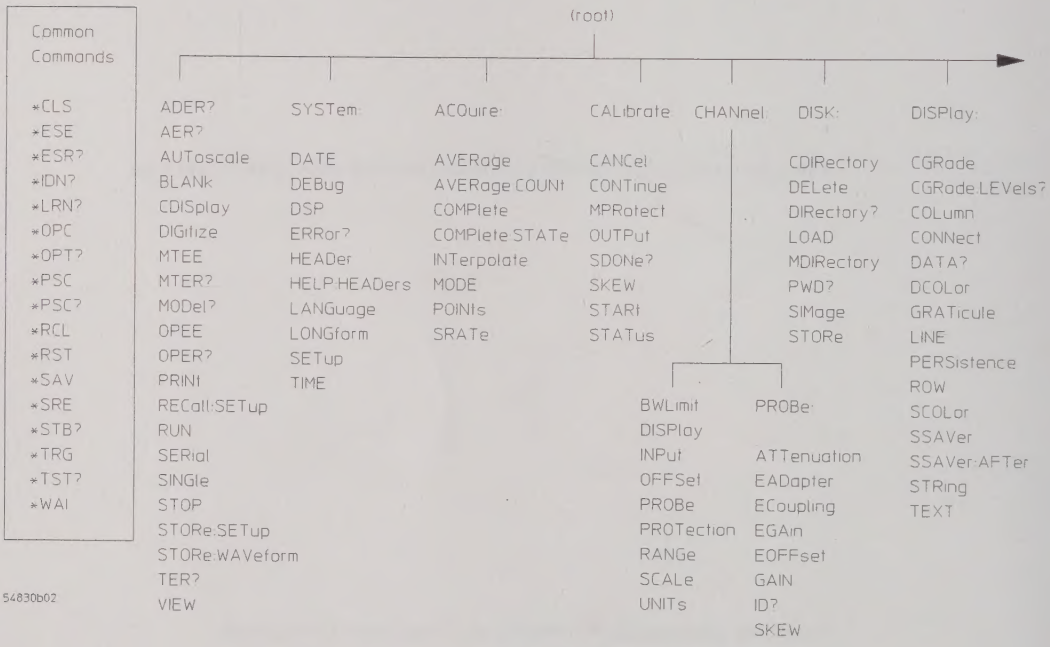
This reference applies directly to software revision code A.04.20 and later.

For Safety information, Warranties, and Regulatory information, see the pages at the end of this book.

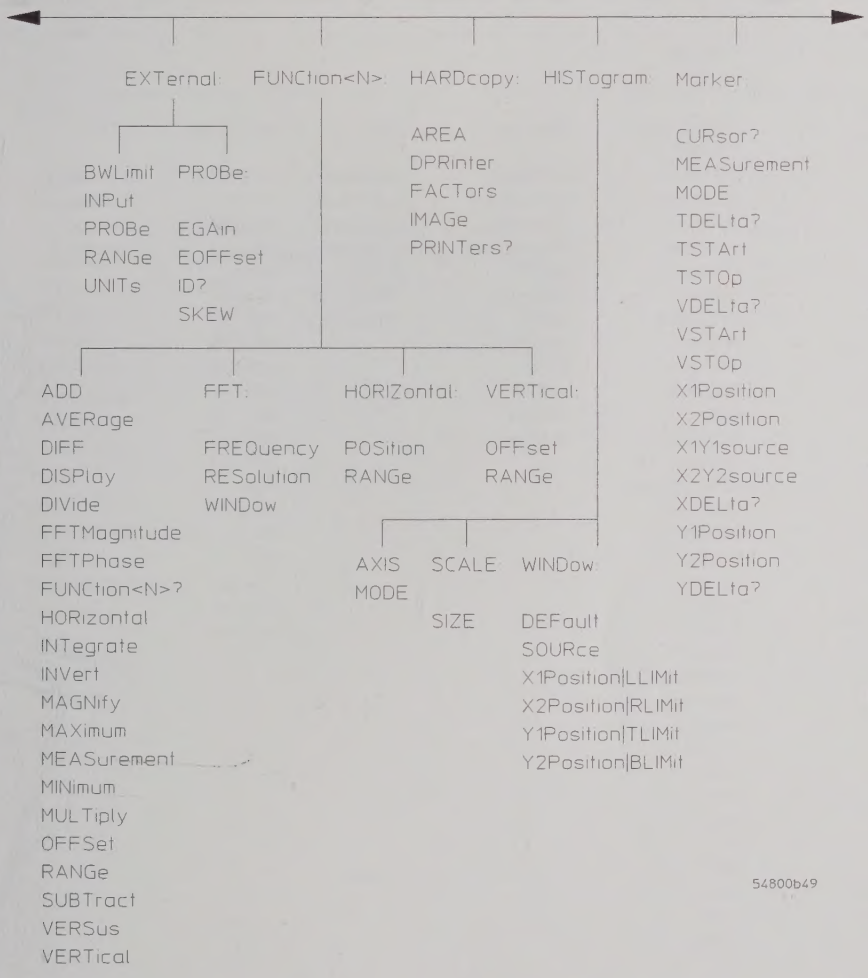
© Copyright Agilent Technologies 1997-2002
All Rights Reserved.

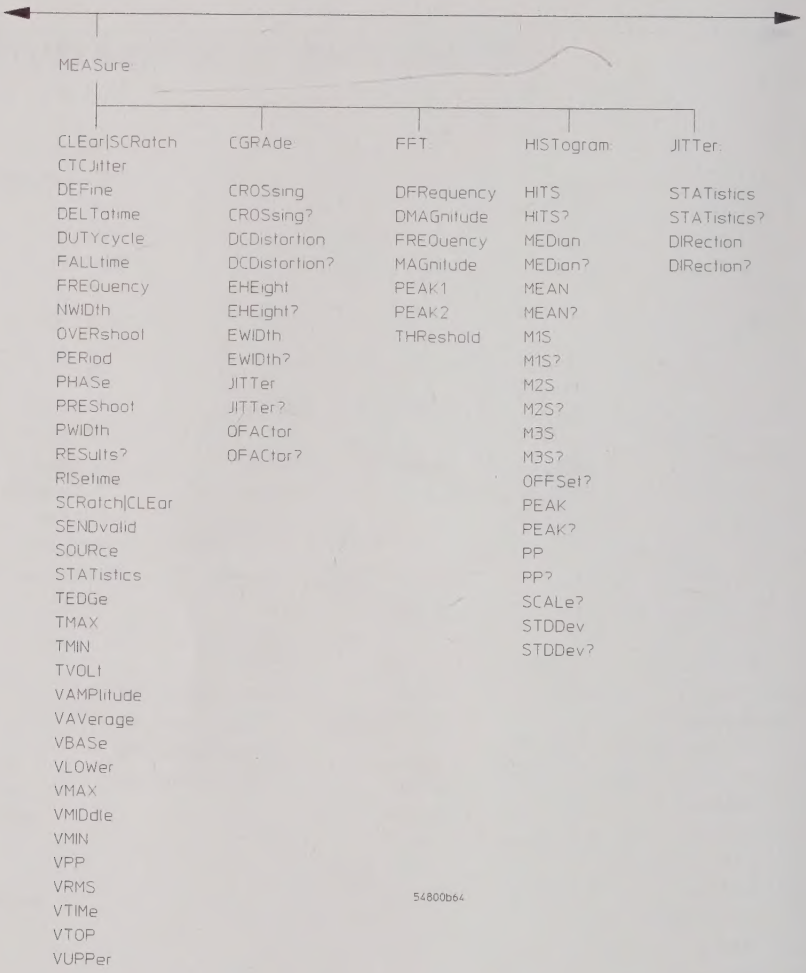
# Infiniium Oscilloscopes

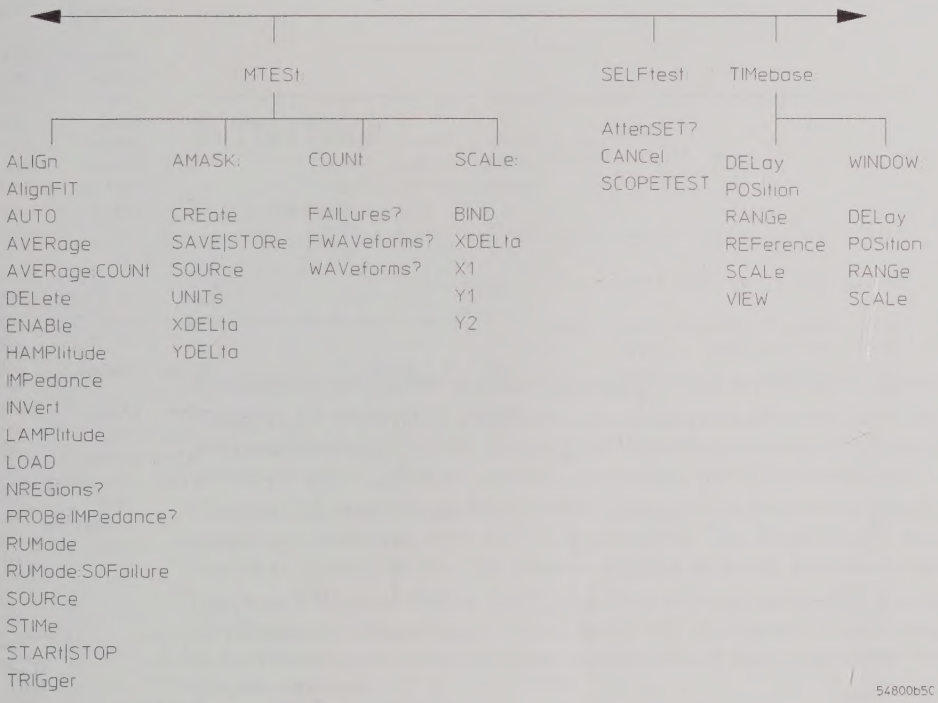
# Programming Command Set

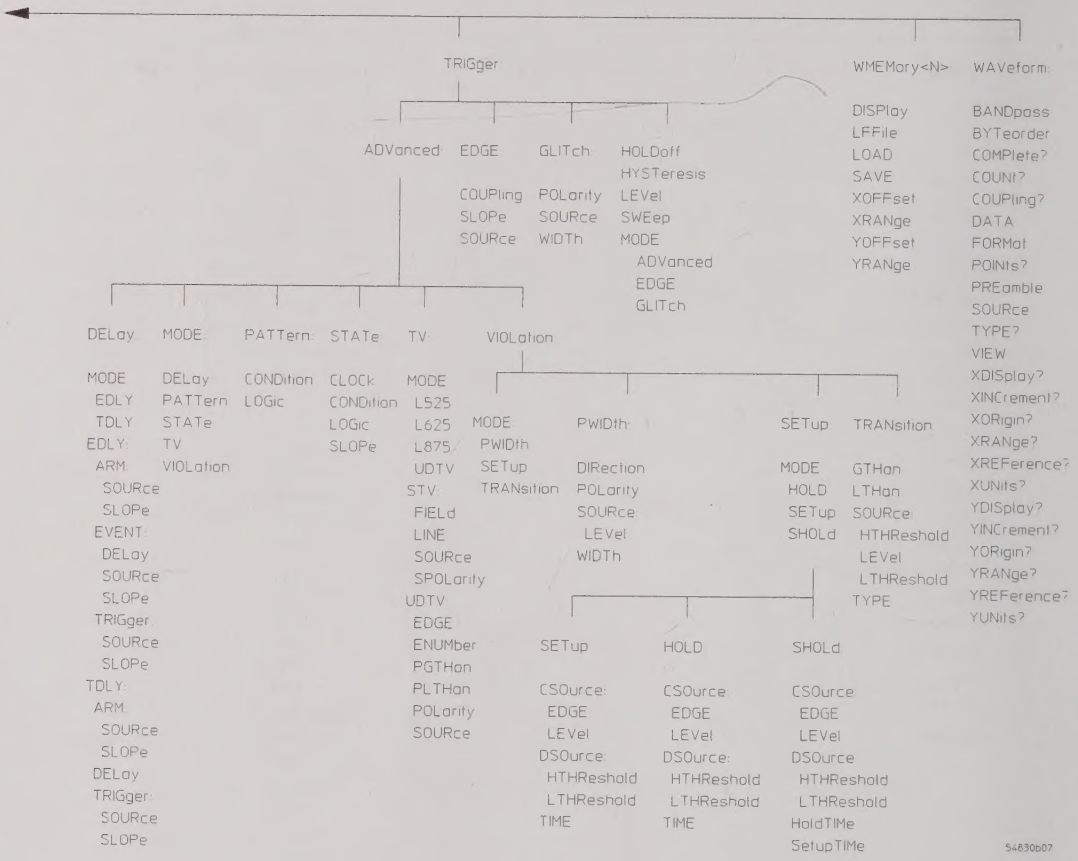



54830b02


EXTernal:
FUNCtion<N>:
HARDcopy:
HISTogram:
Marker
BWLimit
INPut
PROBe
RANGe
UNITs
PROBe:
EGAin
EOFFset
ID?
SKEW
AREA
DPRinter
FACTors
IMAGe
PRINTers?
CURsor?
MEASurement
MODE
TDELta?
TSTArt
TSTOp
VDELta?
VSTArt
VSTOp
X1Position
X2Position
X1Y1source
X2Y2source
XDELta?
Y1Position
Y2Position
YDELta?
ADD
AVERage
DIFF
DISPlay
DIVide
FFTMagnitude
FFTPhase
FUNCtion<N>?
HORizontal
INTegrate
INVert
MAGNify
MAXimum
MEASurement
MINimum
MULTiply
OFFSet
RANGe
SUBTract
VERSus
VERTical
FFT:
FREQuency
RESolution
WINDow
HORIZontal:
POSition
RANGe
VERTical:
OFFset
RANGe
AXIS
MODE
SCALE:
SIZE
WINDow:
DEFault
SOURce
X1Position|LLIMit
X2Position|RLIMit
Y1Position|TLIMit
Y2Position|BLIMit
54800b49


MEASure
CLEar|SCRatch
CTCJitter
DEFine
DELTatime
DUTYcycle
FALLtime
FREQuency
NWIDth
OVERshoot
PERiod
PHASe
PREShoot
PWIDth
RESults?
RISetime
SCRatch|CLEar
SENDvalid
SOURce
STATistics
TEDGe
TMAX
TMIN
TVOLt
VAMPlitude
VAVerage
VBASe
VLOWer
VMAX
VMIDdle
VMIN
VPP
VRMS
VTIMe
VTOP
VUPPer
CGRAde
CROSsing
CROSsing?
DCDistortion
DCDistortion?
EHEight
EHEight?
EWIDth
EWIDth?
JITTer
JITTer?
OFACtor
OFACtor?
FFT
DFRequency
DMAGnitude
FREQuency
MAGnitude
PEAK1
PEAK2
THReshold
HISTogram
HITS
HITS?
MEDian
MEDian?
MEAN
MEAN?
M1S
M1S?
M2S
M2S?
M3S
M3S?
OFFSet?
PEAK
PEAK?
PP
PP?
SCALe?
STDDev
STDDev?
JITTer
STATistics
STATistics?
DIRection
DIRection?
54800b64


MTESt:
ALIGn
AlignFIT
AUTO
AVERage
AVERage:COUNt
DELete
ENABle
HAMPlitude
IMPedance
INVert
LAMPlitude
LOAD
NREGions?
PROBe:IMPedance?
RUMode
RUMode:SOFailure
SOURce
STIMe
STARt|STOP
TRIGger
AMASK:
CREate
SAVE|STORe
SOURce
UNITs
XDELta
YDELta
COUNt
FAILures?
FWAVeforms?
WAVeforms?
SCALe:
BIND
XDELta
X1
Y1
Y2
SELFtest:
AttenSET?
CANCel
SCOPETEST
TIMebase:
DELay
POSition
RANGe
REFerence
SCALe
VIEW
WINDOW:
DELay
POSition
RANGe
SCALe
54800b5C


TRIGger
ADVanced
EDGE
GLITch
HOLDoff
HYSTeresis
LEVel
SWEep
MODE
ADVanced
EDGE
GLITch
COUPling
POLarity
SLOPe
SOURce
WIDTh
DELay
MODE
PATTern
STATe
TV
VIOLation
MODE
DELay
CONDition
LOGic
CLOCk
MODE
L525
L625
L875
UDTV
STV
FIELd
LINE
SOURce
SPOLarity
UDTV
EDGE
ENUMber
PGTHan
PLTHan
POLarity
SOURce
MODE
PWIDth
SETup
TRANsition
PWIDth
DIRection
POLarity
SOURce
LEVel
WIDTh
SETup
TRANsition
MODE
HOLD
SETup
SHOLd
GTHan
LTHan
SOURce
HTHReshold
LEVel
LTHReshold
TYPE
SETup
HOLD
SHOLd
CSOurce
EDGE
LEVel
DSOurce
HTHReshold
LTHReshold
TIME
HoldTIMe
SetupTIMe
WMEMory<N>
DISPlay
LFFile
LOAD
SAVE
XOFFset
XRANge
YOFFset
YRANge
WAVeform
BANDpass
BYTeorder
COMPlete?
COUNt?
COUPling?
DATA
FORMat
POINts?
PREamble
SOURce
TYPE?
VIEW
XDISplay?
XINCrement?
XORigin?
XRANge?
XREFerence?
XUNits?
YDISplay?
YINCrement?
YORigin?
YRANge?
YREFerence?
YUNits?
54830b07

# In This Book

This book is your guide to programming the Infiniium-Series Oscilloscopes.

Chapters 1-5 give you an introduction to programming the oscilloscopes, along with necessary conceptual information. These chapters describe basic program communications, interface, syntax, data types, and status reporting.

Chapter 6 shows example BASIC and C programs, and describes chunks of one program to show you some typical applications. The BASIC and C example programs, and a "readme" file are also shipped on a disk with the oscilloscope.

Chapters 7-24 describe the commands used to program the Infiniium-Series Oscilloscopes. Each chapter describes the set of commands that belong to an individual subsystem, and explains the function of each command. These chapters include:

| | |
|---|---|
| Common | HARDcopy |
| Root Level | HISTogram |
| SYSTem | MARKer |
| ACQuire | MEASure |
| CALibration | Mask TESt |
| CHANnel | SELF-Test |
| DISK | TIMe Base |
| DISPlay | TRIGger |
| EXTernal Channel | WAVeform |
| FUNCtion | Waveform MEMory |

Chapters 25 and 26 describe the language compatibility for 548xx, Hewlett-Packard 547xx, and Hewlett-Packard 545xx Oscilloscopes. These chapters also show you how to choose one of these command languages if you want to use existing programs on Infiniium Oscilloscopes without having to modify your programs.

Chapter 27 describes error messages.

Command syntax diagrams are in the Programmer's Quick Reference.
A Quick Start Guide, Online User's Guide, and Service Guide also ship with the 548xx oscilloscopes.

# Contents

## 1 Introduction to Programming

Communicating with the Oscilloscope 1-3
Output Command 1-4
Device Address 1-4
Instructions 1-4
Instruction Header 1-4
White Space (Separator) 1-5
Braces 1-5
Ellipsis 1-5
Square Brackets 1-5
Program Data 1-5
Header Types 1-6
Duplicate Mnemonics 1-8
Query Headers 1-9
Program Header Options 1-10
Character Program Data 1-10
Numeric Program Data 1-11
Embedded Strings 1-12
Program Message Terminator 1-12
Common Commands within a Subsystem 1-13
Selecting Multiple Subsystems 1-13
Programming Getting Started 1-13
Initialization 1-14
Example Program using HP Basic 1-15
Using the DIGITIZE Command 1-16
Receiving Information from the Oscilloscope 1-18
String Variable Example 1-19
Numeric Variable Example 1-19
Definite-Length Block Response Data 1-20
Multiple Queries 1-21
Oscilloscope Status 1-21

## 2 LAN and GPIB Interfaces

LAN Interface Connector 2-3
GPIB Interface Connector 2-3
Default Startup Conditions 2-4
Interface Capabilities 2-5
GPIB Command and Data Concepts 2-6
Communicating Over the GPIB Interface 2-7
Communicating Over the LAN Interface 2-8
Bus Commands 2-9

**3 Message Communication and System Functions**

Protocols 3-3

**4 Status Reporting**

Status Reporting Data Structures 4-5
Status Byte Register 4-8
Service Request Enable Register 4-10
Message Event Register 4-10
Trigger Event Register 4-10
Standard Event Status Register 4-11
Standard Event Status Enable Register 4-12
Operation Status Register 4-13
Operation Status Enable Register 4-14
Mask Test Event Register 4-15
Mask Test Event Enable Register 4-16
Trigger Armed Event Register 4-17
Error Queue 4-17
Output Queue 4-17
Message Queue 4-18
Clearing Registers and Queues 4-18

**5 Programming Conventions**

Data Flow 5-3
Truncation Rule 5-5
The Command Tree 5-6
Infinity Representation 5-13
Sequential and Overlapped Commands 5-13
Response Generation 5-13
EOI 5-13

**6 Sample Programs**

Sample Program Structure 6-3
Sample C Programs 6-4

Listings of the Sample Programs 6-20
gpibdecl.h Sample Header 6-21
srqagi.c Sample Program 6-23
learnstr.c Sample Program 6-25
sicl_IO.c Sample Program 6-29
natl_IO.c Sample Program 6-33
init.bas Sample Program 6-38

srq.bas Sample Program 6-46
lrn_str.bas Sample Program 6-50

## 7 Common Commands

*CLS (Clear Status) 7-4
*ESE (Event Status Enable) 7-5
*ESR? (Event Status Register) 7-7
*IDN? (Identification Number) 7-9
*LRN? (Learn) 7-10
*OPC (Operation Complete) 7-12
*OPT? (Option) 7-13
*PSC (Power-on Status Clear) 7-14
*RCL (Recall) 7-15
*RST (Reset) 7-16
*SAV (Save) 7-17
*SRE (Service Request Enable) 7-18
*STB? (Status Byte) 7-20
*TRG (Trigger) 7-22
*TST? (Test) 7-23
*WAI (Wait) 7-24

## 8 Root Level Commands

AER? (Arm Event Register) 8-3
AUToscale 8-4
BLANk 8-5
CDISplay 8-6
DIGitize 8-7
MTEE 8-8
MTER? 8-9
MODel? 8-10
OPEE 8-11
OPER? 8-12
OVLEnable 8-13
OVLRegister? 8-14
PRINt 8-15
RECall:SETup 8-16
RUN 8-17
SERial (Serial Number) 8-18
SINGle 8-19
STOP 8-20
STORe:SETup 8-21
STORe:WAVeform 8-22

TER? (Trigger Event Register) 8-23
VIEW 8-24

**9 System Commands**

DATE 9-3
DEBug 9-4
DSP 9-6
ERRor? 9-7
HEADer 9-8
HELP:HEADers 9-10
LANGuage 9-12
LONGform 9-13
SETup 9-15
TIME 9-17

**10 Acquire Commands**

AllowMaxSR 10-3
AVERage 10-4
AVERage:COUNt 10-5
BWLimit 10-6
COMPlete 10-7
COMPlete:STATe 10-9
CONFig 10-10
INTerpolate 10-11
MODE 10-12
POINts 10-13
POINts:AUTO 10-15
SRATe (Sample RATe) 10-16
SRATe:AUTO 10-18

**11 Calibration Commands**

Oscilloscope Calibration 11-3
Probe Calibration 11-4

Calibration Commands 11-5
CANCel 11-6
CONTinue 11-7
MPRotect 11-8
OUTPut 11-9
SDONe? 11-10
SKEW 11-11
STARt 11-12

STATus? 11-13

**12 Channel Commands**

BWLimit 12-3
DISPlay 12-4
INPut 12-5
OFFSet 12-6
PROBe 12-7
PROBe:ATTenuation 12-9
PROBe:EADapter 12-10
PROBe:ECoupling 12-12
PROBe:EGAin 12-14
PROBe:EOFFset 12-15
PROBe:GAIN 12-16
PROBe:ID? 12-17
PROBe:SKEW 12-18
PROTection:CLEar 12-19
PROTection? 12-20
RANGe 12-21
SCALe 12-22
UNITs 12-23

**13 Disk Commands**

CDIRectory 13-3
DELete 13-4
DIRectory? 13-5
LOAD 13-6
MDIRectory 13-7
PWD? 13-8
SIMage 13-9
STORe 13-10

**14 Display Commands**

CGRade 14-3
CGRade:LEVels? 14-5
COLumn 14-7
CONNect 14-8
DATA? 14-9
DCOLor 14-10
GRATicule 14-11
LINE 14-13
PERSistence 14-14

ROW 14-15
SCOLor 14-16
SSAVer 14-19
STRing 14-20
TEXT 14-21

## 15 External Channel Commands

BWLimit 15-3
INPut 15-4
PROBe 15-5
PROBe:ATTenuation 15-6
PROBe:EADapter 15-7
PROBe:ECoupling 15-9
PROBe:EGAin 15-11
PROBe:EOFFset 15-12
PROBe:GAIN 15-13
PROBe:ID? 15-14
PROBe:SKEW 15-15
RANGe 15-16
UNITs 15-17

## 16 Function Commands

FUNCtion<N>? 16-4
ADD 16-5
AVERage 16-6
DIFF (Differentiate) 16-7
DISPlay 16-8
DIVide 16-9
FFT:FREQuency 16-10
FFT:RESolution? 16-11
FFT:WINDow 16-12
FFTMagnitude 16-14
FFTPhase 16-15
HORizontal 16-16
HORizontal:POSition 16-17
HORizontal:RANGe 16-18
INTegrate 16-19
INVert 16-20
MAGNify 16-21
MAXimum 16-22
MEASurement 16-23
MINimum 16-25

MULTiply 16-26
OFFSet 16-27
RANGe 16-28
SUBTract 16-29
VERSus 16-30
VERTical 16-31
VERTical:OFFSet 16-32
VERTical:RANGe 16-33

**17 Hardcopy Commands**

AREA 17-3
DPRinter 17-4
FACTors 17-6
IMAGe 17-7
PRINters? 17-8

**18 Histogram Commands**

AXIS 18-4
MODE 18-5
SCALe:SIZE 18-6
WINDow:DEFault 18-7
WINDow:SOURce 18-8
WINDow:X1Position | LLIMit 18-9
WINDow:X2Position | RLIMit 18-10
WINDow:Y1Position | BLIMit 18-11
WINDow:Y2Position | TLIMit 18-12

**19 Marker Commands**

CURSor? 19-3
MEASurement:READout 19-4
MODE 19-5
TDELta? 19-6
TSTArt 19-7
TSTOp 19-9
VDELta? 19-11
VSTArt 19-12
VSTOp 19-14
X1Position 19-16
X2Position 19-17
X1Y1source 19-18
X2Y2source 19-19
XDELta? 19-20

Y1Position 19-21
Y2Position 19-22
YDELta? 19-23

## 20 Measure Commands

AREA 20-6
CGRade:CROSsing 20-7
CGRade:DCDistortion 20-8
CGRade:EHEight 20-9
CGRade:EWIDth 20-10
CGRade:JITTer 20-11
CGRade:QFACtor 20-12
CLEar 20-13
CTCJitter 20-14
DEFine 20-16
DELTatime 20-20
DUTYcycle 20-22
FALLtime 20-24
FFT:DFRequency 20-26
FFT:DMAGnitude 20-27
FFT:FREQuency 20-28
FFT:MAGNitude 20-29
FFT:PEAK1 20-30
FFT:PEAK2 20-31
FFT:THReshold 20-32
FREQuency 20-33
HISTogram:HITS 20-35
HISTogram:MEAN 20-37
HISTogram:MEDian 20-39
HISTogram:M1S 20-41
HISTogram:M2S 20-43
HISTogram:M3S 20-45
HISTogram:PEAK 20-47
HISTogram:PP 20-49
HISTogram:STDDev 20-51
JITTer:DIRection 20-53
JITTer:STATistics 20-55
NWIDth 20-57
OVERshoot 20-59
PERiod 20-61
PHASe 20-63
PREShoot 20-65
PWIDth 20-67

RESults? 20-69
RISetime 20-72
SCRatch 20-74
SENDvalid 20-75
SOURce 20-76
STATistics 20-77
TEDGe 20-78
TMAX 20-80
TMIN 20-82
TVOLt 20-83
VAMPlitude 20-85
VAVerage 20-87
VBASe 20-89
VLOWer 20-91
VMAX 20-92
VMIDdle 20-94
VMIN 20-95
VPP 20-97
VRMS 20-99
VTIMe 20-101
VTOP 20-102
VUPPer 20-104

**21 Mask Test Commands**

ALIGn 21-4
AlignFIT 21-5
AMASk:CREate 21-7
AMASk:SOURce 21-8
AMASk:SAVE | STORe 21-9
AMASk:UNITs 21-10
AMASk:XDELta 21-11
AMASk:YDELta 21-13
AUTO 21-15
AVERage 21-16
AVERage:COUNt 21-17
COUNt:FAILures? 21-18
COUNt:FWAVeforms? 21-19
COUNt:WAVeforms? 21-20
DELete 21-21
ENABle 21-22
HAMPlitude 21-23
IMPedance 21-24
INVert 21-26

LAMPlitude 21-27
LOAD 21-28
NREGions? 21-29
PROBe:IMPedance? 21-30
RUMode 21-31
RUMode:SOFailure 21-33
SCALe:BIND 21-34
SCALe:X1 21-35
SCALe:XDELta 21-36
SCALe:Y1 21-37
SCALe:Y2 21-38
SOURce 21-39
STARt | STOP 21-40
STIMe 21-41
TITLe? 21-42
TRIGger:SOURce 21-43

**22 Self-Test Commands**

AttenSET? 22-3
CANCel 22-4
SCOPETEST 22-5

**23 Time Base Commands**

DELay 23-3
POSition 23-5
RANGe 23-6
REFerence 23-7
SCALe 23-8
VIEW 23-9
WINDow:DELay 23-10
WINDow:POSition 23-12
WINDow:RANGe 23-13
WINDow:SCALe 23-14

**24 Trigger Commands**

Organization of Trigger Modes and Commands 24-5

Summary of Trigger Modes and Commands 24-6
Trigger Modes 24-8
HOLDoff 24-9
HTHReshold 24-10
HYSTeresis 24-11

LEVel 24-12
LTHReshold 24-13
SWEep 24-14

Edge Trigger Mode and Commands 24-15
EDGE:COUPling 24-17
EDGE:SLOPe 24-18
EDGE:SOURce 24-19

Glitch Trigger Mode and Commands 24-20
GLITch:POLarity 24-22
GLITch:SOURce 24-23
GLITch:WIDTh 24-24

Advanced COMM Trigger Mode and Commands 24-25
COMM:BWIDth 24-26
COMM:ENCode 24-27
COMM:LEVel 24-28
COMM:PATTern 24-29
COMM:POLarity 24-30
COMM:SOURce 24-31

Advanced Pattern Trigger Mode and Commands 24-32
PATTern:CONDition 24-34
PATTern:LOGic 24-35

Advanced State Trigger Mode and Commands 24-36
STATe:CLOCk 24-38
STATe:CONDition 24-39
STATe:LOGic 24-40
STATe:LTYPe 24-41
STATe:SLOPe 24-42

Advanced Delay By Event Mode and Commands 24-43
EDLY:ARM:SOURce 24-45
EDLY:ARM:SLOPe 24-46
EDLY:EVENt:DELay 24-47
EDLY:EVENt:SOURce 24-48
EDLY:EVENt:SLOPe 24-49
EDLY:TRIGger:SOURce 24-50
EDLY:TRIGger:SLOPe 24-51

Advanced Delay By Time Mode and Commands 24-52
TDLY:ARM:SOURce 24-54
TDLY:ARM:SLOPe 24-55
TDLY:DELay 24-56
TDLY:TRIGger:SOURce 24-57
TDLY:TRIGger:SLOPe 24-58

Advanced Standard TV Mode and Commands 24-59
STV:FIELd 24-61
STV:LINE 24-62
STV:SOURce 24-63
STV:SPOLarity 24-64

Advanced User Defined TV Mode and Commands 24-65
UDTV:EDGE 24-68
UDTV:ENUMber 24-69
UDTV:PGTHan 24-70
UDTV:PLTHan 24-71
UDTV:POLarity 24-72
UDTV:SOURce 24-73

Advanced Trigger Violation Modes 24-75
VIOLation:MODE 24-76

Pulse Width Violation Mode and Commands 24-77
VIOLation:PWIDth:SOURce 24-79
VIOLation:PWIDth:POLarity 24-80
VIOLation:PWIDth:DIRection 24-81
VIOLation:PWIDth:WIDTh 24-82

Setup Violation Mode and Commands 24-83
VIOLation:SETup:MODE 24-86
VIOLation:SETup:SETup:CSOurce 24-87
VIOLation:SETup:SETup:CSOurce:LEVel 24-88
VIOLation:SETup:SETup:CSOurce:EDGE 24-89
VIOLation:SETup:SETup:DSOurce 24-90
VIOLation:SETup:SETup:DSOurce:HTHReshold 24-91
VIOLation:SETup:SETup:DSOurce:LTHReshold 24-92
VIOLation:SETup:SETup:TIME 24-93
VIOLation:SETup:HOLD:CSOurce 24-94
VIOLation:SETup:HOLD:CSOurce:LEVel 24-95
VIOLation:SETup:HOLD:CSOurce:EDGE 24-96

VIOLation:SETup:HOLD:DSOurce 24-97
VIOLation:SETup:HOLD:DSOurce:HTHReshold 24-98
VIOLation:SETup:HOLD:DSOurce:LTHReshold 24-99
VIOLation:SETup:HOLD:TIME 24-100
VIOLation:SETup:SHOLd:CSOurce 24-101
VIOLation:SETup:SHOLd:CSOurce:LEVel 24-102
VIOLation:SETup:SHOLd:CSOurce:EDGE 24-103
VIOLation:SETup:SHOLd:DSOurce 24-104
VIOLation:SETup:SHOLd:DSOurce:HTHReshold 24-105
VIOLation:SETup:SHOLd:DSOurce:LTHReshold 24-106
VIOLation:SETup:SHOLd:SetupTIMe (STIMe) 24-107
VIOLation:SETup:SHOLd:HoldTIMe (HTIMe) 24-108

Transition Violation Mode 24-109
VIOLation:TRANsition 24-111
VIOLation:TRANsition:SOURce 24-112
VIOLation:TRANsition:SOURce:HTHReshold 24-113
VIOLation:TRANsition:SOURce:LTHReshold 24-114
VIOLation:TRANsition:TYPE 24-115

**25 Waveform Commands**

BANDpass? 25-5
BYTeorder 25-6
CLIPped? 25-7
COMPlete? 25-8
COUNt? 25-9
COUPling? 25-10
DATA 25-11
FORMat 25-32
POINts? 25-35
PREamble 25-36
SOURce 25-41
TYPE? 25-42
VIEW 25-44
XDISplay? 25-46
XINCrement? 25-47
XORigin? 25-48
XRANge? 25-49
XREFerence? 25-50
XUNits? 25-51
YDISplay? 25-52
YINCrement? 25-53
YORigin? 25-54

YRANge? 25-55
YREFerence? 25-56
YUNits? 25-57

## 26 Waveform Memory Commands

DISPlay 26-3
LOAD 26-4
SAVE 26-5
XOFFset 26-6
XRANge 26-7
YOFFset 26-8
YRANge 26-9

## 27 Infiniium and HP 547XX Digitizing Oscilloscopes Language Compatibility

To select a command language 27-4
Acquisition System Command Language Compatibility 27-5
Calibration Command Language Compatibility 27-6
Channel Command Language Compatibility 27-7
Disk Command Language Compatibility 27-8
Display Command Language Compatibility 27-9
External Command Language Compatibility 27-10
FFT Command Language Compatibility 27-11
Function Command Language Compatibility 27-12
Hardcopy Command Language Compatibility 27-13
Limit Test Command Language Compatibility 27-14
Marker Command Language Compatibility 27-15
Measure Command Language Compatibility 27-16
Multiple Memory Command Language Compatibility 27-17
Memory Test Command Language Compatibility 27-18
Pixel Memory Command Language Compatibility 27-19
Self-Test Command Language Compatibility 27-20
Sequential Command Language Compatibility 27-21
System Command Language Compatibility 27-22
Time Base Command Language Compatibility 27-23
Trigger Command Language Compatibility 27-24
Waveform Command Language Compatibility 27-27
Waveform Memory Command Language Compatibility 27-28
Root Command Language Compatibility 27-29
Common Command Language Compatibility 27-30

## 28 Infiniium and HP 545XX Oscilloscopes Language Compatibility

To select a command language 28-4
Acquisition System Command Language Compatibility 28-5
Calibration Command Language Compatibility 28-6
Channel Command Language Compatibility 28-7
Disk Command Language Compatibility 28-8
Display Command Language Compatibility 28-9
External Command Language Compatibility 28-10
FFT Command Language Compatibility 28-11
Function Command Language Compatibility 28-12
Hardcopy Command Language Compatibility 28-13
Limit Test Command Language Compatibility 28-14
Marker Command Language Compatibility 28-15
Measure Command Language Compatibility 28-16
Multiple Memory Command Language Compatibility 28-18
Memory Test Command Language Compatibility 28-19
Pixel Memory Command Language Compatibility 28-20
Self-Test Command Language Compatibility 28-21
Sequential Command Language Compatibility 28-22
System Command Language Compatibility 28-23
Time Base Command Language Compatibility 28-24
Trigger Command Language Compatibility 28-25
Waveform Command Language Compatibility 28-28
Waveform Memory Command Language Compatibility 28-29
Root Command Language Compatibility 28-30
Common Command Language Compatibility 28-31

## 29 Error Messages

Error Queue 29-3
Error Numbers 29-4
Command Error 29-5
Execution Error 29-6
Device- or Oscilloscope-Specific Error 29-7
Query Error 29-8
List of Error Messages 29-9

# 1

# Introduction to Programming

# Introduction to Programming

This chapter introduces the basics for remote programming of an oscilloscope. The programming commands in this manual conform to the IEEE 488.2 Standard Digital Interface for Programmable Instrumentation. The programming commands provide the means of remote control.

Basic operations that you can do with a computer and an oscilloscope include:

- Set up the oscilloscope.
- Make measurements.
- Get data (waveform, measurements, and configuration) from the oscilloscope.
- Send information, such as waveforms and configurations, to the oscilloscope.

You can accomplish other tasks by combining these functions.

**Example Programs are Written in HP BASIC and C**

The programming examples for individual commands in this manual are written in HP BASIC and C.

# Communicating with the Oscilloscope

Computers communicate with the oscilloscope by sending and receiving messages over a remote interface, such as a GPIB card or a Local Area Network (LAN) card. Commands for programming normally appear as ASCII character strings embedded inside the output statements of a "host" language available on your computer. The input commands of the host language are used to read responses from the oscilloscope.

For example, HP BASIC uses the OUTPUT statement for sending commands and queries. After a query is sent, the response is usually read using the HP BASIC ENTER statement. The ENTER statement passes the value across the bus to the computer and places it in the designated variable.

For the GPIB interface, messages are placed on the bus using an output command and passing the device address, program message, and a terminator. Passing the device address ensures that the program message is sent to the correct GPIB interface and GPIB device.

The following HP BASIC OUTPUT statement sends a command that sets the channel 1 scale value to 500 mV:

```
OUTPUT <device address> ;":CHANNEL1:SCALE 500E-
3"<terminator>
```

The device address represents the address of the device being programmed. Each of the other parts of the above statement are explained on the following pages.

**Use the Suffix Multiplier Instead**

Using "mV" or "V" following the numeric voltage value in some commands will cause Error 138 - Suffix not allowed. Instead, use the convention for the suffix multiplier as described in chapter 3, "Message Communication and System Functions."

## Output Command

The output command depends entirely on the programming language. Throughout this book, HP BASIC and ANSI C are used in the examples of individual commands. If you are using other languages, you will need to find the equivalents of HP BASIC commands like OUTPUT, ENTER, and CLEAR, to convert the examples.

## Device Address

The location where the device address must be specified depends on the programming language you are using. In some languages, it may be specified outside the OUTPUT command. In HP BASIC, it is always specified after the keyword, OUTPUT. The examples in this manual assume that the oscilloscope and interface card are at GPIB device address 707. When writing programs, the device address varies according to how the bus is configured.

## Instructions

Instructions, both commands and queries, normally appear as strings embedded in a statement of your host language, such as BASIC, Pascal, or C. The only time a parameter is not meant to be expressed as a string is when the instruction's syntax definition specifies <block data>, such as HP BASIC's "learnstring" command. There are only a few instructions that use block data.

Instructions are composed of two main parts:

- The header, which specifies the command or query to be sent.
- The program data, which provides additional information to clarify the meaning of the instruction.

## Instruction Header

The instruction header is one or more command mnemonics separated by colons (:). They represent the operation to be performed by the oscilloscope. See the "Programming Conventions" chapter for more information.

Queries are formed by adding a question mark (?) to the end of the header. Many instructions can be used as either commands or queries, depending on whether or not you include the question mark. The command and query forms of an instruction usually have different program data. Many queries do not use any program data.

## White Space (Separator)

White space is used to separate the instruction header from the program data. If the instruction does not require any program data parameters, you do not need to include any white space. In this manual, white space is defined as one or more spaces. ASCII defines a space to be character 32 in decimal.

## Braces

When several items are enclosed by braces, { }, only one of these elements may be selected. Vertical line ( | ) indicates "or". For example, {ON | OFF} indicates that only ON or OFF may be selected, not both.

## Ellipsis

... An ellipsis (trailing dots) indicates that the preceding element may be repeated one or more times.

## Square Brackets

Items enclosed in square brackets, [ ], are optional.

## Program Data

Program data is used to clarify the meaning of the command or query. It provides necessary information, such as whether a function should be on or off, or which waveform is to be displayed. Each instruction's syntax definition shows the program data and the values they accept. See the Programmer's Quick Reference Guide for more information about general syntax rules and acceptable values.

When there is more than one data parameter, they are separated by commas (,). You can add spaces around the commas to improve readability.

# Header Types

There are three types of headers:

- Simple Command headers
- Compound Command headers
- Common Command headers

### Simple Command Header

Simple command headers contain a single mnemonic. AUTOSCALE and DIGITIZE are examples of simple command headers typically used in this oscilloscope. The syntax is:

```
<program mnemonic><terminator>
```

or

```
OUTPUT 707;":AUTOSCALE"
```

When program data must be included with the simple command header (for example, :DIGITIZE CHAN1), white space is added to separate the data from the header. The syntax is:

```
<program mnemonic><separator><program data><terminator>
```

or

```
OUTPUT 707;":DIGITIZE CHANNEL1,FUNCTION2"
```

### Compound Command Header

Compound command headers are a combination of two program mnemonics. The first mnemonic selects the subsystem, and the second mnemonic selects the function within that subsystem. The mnemonics within the compound message are separated by colons. For example:

To execute a single function within a subsystem:

```
:<subsystem>:<function><separator><program data><terminator>
```

For example:

```
OUTPUT 707;":CHANNEL1:BWLIMIT ON"
```

### Combining Commands in the Same Subsystem

To execute more than one command within the same subsystem, use a semicolon (;) to separate the commands:

```
:<subsystem>:<command><separator><data>;<command><separator>
<data><terminator>
```

For example:

```
:CHANNEL1:INPUT DC;BWLIMIT ON
```

### Common Command Header

Common command headers, such as clear status, control the IEEE 488.2 functions within the oscilloscope. The syntax is:

```
*<command header><terminator>
```

No space or separator is allowed between the asterisk (*) and the command header. *CLS is an example of a common command header.

## Duplicate Mnemonics

Identical function mnemonics can be used for more than one subsystem. For example, you can use the function mnemonic RANGE to change both the vertical range and horizontal range:

To set the vertical range of channel 1 to 0.4 volts full scale:

```
:CHANNEL1:RANGE .4
```

To set the horizontal time base to 1 second full scale:

```
:TIMEBASE:RANGE 1
```

In these examples, CHANNEL1 and TIMEBASE are subsystem selectors, and determine the range type being modified.

## Query Headers

A command header immediately followed by a question mark (?) is a query. After receiving a query, the oscilloscope interrogates the requested subsystem and places the answer in its output queue. The answer remains in the output queue until it is read or until another command is issued. When read, the answer is transmitted across the bus to the designated listener (typically a computer). For example, the query:

```
:TIMEBASE:RANGE?
```

places the current time base setting in the output queue.

In HP BASIC, the computer input statement:

```
ENTER < device address > ;Range
```

passes the value across the bus to the computer and places it in the variable Range.

You can use queries to find out how the oscilloscope is currently configured and to get results of measurements made by the oscilloscope.
For example, the command:

```
:MEASURE:RISETIME?
```

tells the oscilloscope to measure the rise time of your waveform and place the result in the output queue.

The output queue must be read before the next program message is sent. For example, when you send the query :MEASURE:RISETIME?, you must follow it with an input statement. In HP BASIC, this is usually done with an ENTER statement immediately followed by a variable name. This statement reads the result of the query and places the result in a specified variable.

**Handle Queries Properly**

If you send another command or query before reading the result of a query, the output buffer is cleared and the current response is lost. This also generates a query-interrupted error in the error queue. If you execute an input statement before you send a query, it will cause the computer to wait indefinitely.

## Program Header Options

You can send program headers using any combination of uppercase or lowercase ASCII characters. Oscilloscope responses, however, are always returned in uppercase.

You may send program command and query headers in either long form (complete spelling), short form (abbreviated spelling), or any combination of long form and short form. For example:

:TIMEBASE:DELAY 1E-6 is the long form.

:TIM:DEL 1E-6 is the short form.

**Using Long Form or Short Form**

Programs written in long form are easily read and are almost self-documenting. The short form syntax conserves the amount of computer memory needed for program storage and reduces I/O activity.

The rules for the short form syntax are described in the chapter, "Programming Conventions."

## Character Program Data

Character program data is used to convey parameter information as alpha or alphanumeric strings. For example, the :TIMEBASE:REFERENCE command can be set to left, center, or right. The character program data in this case may be LEFT, CENTER, or RIGHT. The command :TIMEBASE:REFERENCE RIGHT sets the time base reference to right.

The available mnemonics for character program data are always included with the instruction's syntax definition. You may send either the long form of commands, or the short form (if one exists). You may mix uppercase and lowercase letters freely. When receiving responses, uppercase letters are used exclusively.

## Numeric Program Data

Some command headers require program data to be expressed numerically. For example, :TIMEBASE:RANGE requires the desired full-scale range to be expressed numerically.

For numeric program data, you can use exponential notation or suffix multipliers to indicate the numeric value. The following numbers are all equal:

28 = 0.28E2 = 280E-1 = 28000m = 0.028K = 28E-3K

When a syntax definition specifies that a number is an integer, it means that the number should be whole. Any fractional part is ignored and truncated. Numeric data parameters that accept fractional values are called real numbers. For more information see the chapter, "Interface Functions."

All numbers are expected to be strings of ASCII characters.

- When sending the number 9, you would send a byte representing the ASCII code for the character "9" (which is 57).
- A three-digit number like 102 would take up three bytes (ASCII codes 49, 48, and 50). The number of bytes is figured automatically when you include the entire instruction in a string.

## Embedded Strings

Embedded strings contain groups of alphanumeric characters which are treated as a unit of data by the oscilloscope. An example of this is the line of text written to the advisory line of the oscilloscope with the :SYSTEM:DSP command:

```
:SYSTEM:DSP ""This is a message.""
```

You may delimit embedded strings with either single (') or double (") quotation marks. These strings are case-sensitive, and spaces are also legal characters.

## Program Message Terminator

The program instructions within a data message are executed after the program message terminator is received. The terminator may be either an NL (New Line) character, an EOI (End-Or-Identify) asserted in the GPIB interface, or a combination of the two. Asserting the EOI sets the EOI control line low on the last byte of the data message. The NL character is an ASCII linefeed (decimal 10).

**New Line Terminator Functions Like EOS and EOT**

The NL (New Line) terminator has the same function as an EOS (End Of String) and EOT (End Of Text) terminator.

## Common Commands within a Subsystem

Common commands can be received and processed by the oscilloscope whether they are sent over the bus as separate program messages or within other program messages. If you have selected a subsystem, and a common command is received by the oscilloscope, the oscilloscope remains in the selected subsystem. For example, if the program message

```
":ACQUIRE:AVERAGE ON;*CLS;COUNT 1024"
```

is received by the oscilloscope, the oscilloscope turns averaging on, then clears the status information without leaving the selected subsystem.

If some other type of command is received within a program message, you must re-enter the original subsystem after the command. For example, the program message

```
":ACQUIRE:AVERAGE ON;:AUTOSCALE;:ACQUIRE:AVERAGE:COUNT 1024"
```

turns averaging on, completes the autoscale operation, then sets the acquire average count. Here, :ACQUIRE must be sent again after AUTOSCALE to re-enter the ACQUIRE subsystem and set the count.

## Selecting Multiple Subsystems

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon lets you enter a new subsystem. For example:

```
<program mnemonic><data>;:<program mnemonic><data><terminator>
```

:CHANNEL1:RANGE 0.4;:TIMEBASE:RANGE 1

**You can Combine Compound and Simple Commands**

Multiple program commands may be any combination of compound and simple commands.

## Programming Getting Started

The remainder of this chapter explains how to set up the oscilloscope, how to retrieve setup information and measurement results, how to digitize a waveform, and how to pass data to the computer. The chapter, "Measure Commands" describes sending measurement data to the oscilloscope.

# Initialization

To make sure the bus and all appropriate interfaces are in a known state, begin every program with an initialization statement. For example, HP BASIC provides a CLEAR command which clears the interface buffer:

```
CLEAR 707  ! initializes the interface of the oscilloscope
```

When you are using GPIB, CLEAR also resets the oscilloscope's parser. The parser is the program that reads in the instructions you send.

After clearing the interface, initialize the oscilloscope to a preset state:

```
OUTPUT 707;"*RST"  ! initializes the oscilloscope to a preset
state
```

**Initializing the Oscilloscope**

The commands and syntax for initializing the oscilloscope are discussed in the chapter, "Common Commands." Refer to your GPIB manual and programming language reference manual for information on initializing the interface.

### Autoscale

The AUTOSCALE feature of Agilent Technologies digitizing oscilloscopes performs a very useful function on unknown waveforms by automatically setting up the vertical channel, time base, and trigger level of the oscilloscope.

The syntax for the autoscale function is:

```
:AUTOSCALE<terminator>
```

### Setting Up the Oscilloscope

A typical oscilloscope setup configures the vertical range and offset voltage, the horizontal range, delay time, delay reference, trigger mode, trigger level, and slope.

A typical example of the commands sent to the oscilloscope are:

```
:CHANNEL1:PROBE 10; RANGE 16;OFFSET 1.00<terminator>
:SYSTEM:HEADER OFF<terminator>
:TIMEBASE:RANGE 1E-3;DELAY 100E-6<terminator>
```

This example sets the time base at 1 ms full-scale (100 µs/div), with delay of 100 µs. Vertical is set to 16 V full-scale (2 V/div), with center of screen at 1 V, and probe attenuation of 10.

# Example Program using HP Basic

This program demonstrates the basic command structure used to program the oscilloscope.

```
10    CLEAR 707! Initialize oscilloscope interface
20    OUTPUT 707;"*RST"!Initialize oscilloscope to preset state
30    OUTPUT 707;":TIMEBASE:RANGE 5E-4"! Time base to 500 us full scale
40    OUTPUT 707;":TIMEBASE:DELAY 0"! Delay to zero
50    OUTPUT 707;":TIMEBASE:REFERENCE CENTER"! Display reference at center
60    OUTPUT 707;":CHANNEL1:PROBE 10"! Probe attenuation to 10:1
70    OUTPUT 707;":CHANNEL1:RANGE 1.6"! Vertical range to 1.6 V full scale
80    OUTPUT 707;":CHANNEL1:OFFSET -.4"! Offset to -0.4
90    OUTPUT 707;":CHANNEL1:INPUT DC"! Coupling to DC
100   OUTPUT 707;":TRIGGER:MODE EDGE"! Edge triggering
110   OUTPUT 707;":TRIGGER:LEVEL CHAN1,-.4"! Trigger level to -0.4
120   OUTPUT 707;":TRIGGER:SLOPE POSITIVE"! Trigger on positive slope
125   OUTPUT 707;":SYSTEM:HEADER OFF<terminator>
130   OUTPUT 707;":ACQUIRE:MODE RTIME"! Normal acquisition
140   OUTPUT 707;":DISPLAY:GRATICULE FRAME"! Grid off
150   END
```

**Overview of the Program**

- Line 10 initializes the oscilloscope interface to a known state.
- Line 20 initializes the oscilloscope to a preset state.
- Lines 30 through 50 set the time base, the horizontal time at 500 μs full scale, and 0 s of delay referenced at the center of the graticule.
- Lines 60 through 90 set 10:1 probe attenuation, set the vertical range to 1.6 volts full scale, center screen at –0.4 volts, and select DC 1 Mohm impedance coupling.
- Lines 100 through 120 configure the oscilloscope to trigger at –0.4 volts with positive edge triggering.
- Line 125 turns system headers off.
- Line 130 configures the oscilloscope for real time acquisition.
- Line 140 turns the grid off.

# Using the DIGITIZE Command

The DIGITIZE command is a macro that captures data using the acquisition (ACQUIRE) subsystem. When the digitize process is complete, the acquisition is stopped. You can measure the captured data by using the oscilloscope or by transferring the data to a computer for further analysis. The captured data consists of two parts: the preamble and the waveform data record.

After changing the oscilloscope configuration, the waveform buffers are cleared. Before doing a measurement, you should send the DIGITIZE command to ensure new data has been collected.

You can send the DIGITIZE command with no parameters for a higher throughput. Refer to the DIGITIZE command in the chapter, "Root Level Commands" for details.

When the DIGITIZE command is sent to an oscilloscope, the specified channel's waveform is digitized using the current ACQUIRE parameters. Before sending the :WAVEFORM:DATA? query to download waveform data to your computer, you should specify the WAVEFORM parameters.

The number of data points comprising a waveform varies according to the number requested in the ACQUIRE subsystem. The ACQUIRE subsystem determines the number of data points, type of acquisition, and number of averages used by the DIGITIZE command. This lets you specify exactly what the digitized information contains. The following program example shows a typical setup:

```
OUTPUT 707;":SYSTEM:HEADER OFF<terminator>
OUTPUT 707;":ACQUIRE:MODE RTIME"<terminator>
OUTPUT 707;":ACQUIRE:COMPLETE 100"<terminator>
OUTPUT 707;":WAVEFORM:SOURCE CHANNEL1"<terminator>
OUTPUT 707;":WAVEFORM:FORMAT BYTE"<terminator>
OUTPUT 707;":ACQUIRE:COUNT 8"<terminator>
OUTPUT 707;":ACQUIRE:POINTS 500"<terminator>
OUTPUT 707;":DIGITIZE CHANNEL1"<terminator>
OUTPUT 707;":WAVEFORM:DATA?"<terminator>
```

This setup places the oscilloscope into the real time sampling mode using eight averages. This means that when the DIGITIZE command is received, the command will execute until the waveform has been averaged at least eight times.

After receiving the :WAVEFORM:DATA? query, the oscilloscope will start downloading the waveform information.

Digitized waveforms are passed from the oscilloscope to the computer by sending a numerical representation of each digitized point. The format of the numerical representation is controlled by using the :WAVEFORM:FORMAT command and may be selected as BYTE, WORD, or ASCII.

The easiest method of receiving a digitized waveform depends on data structures, available formatting, and I/O capabilities. You must convert the data values to determine the voltage value of each point. These data values are passed starting with the left most point on the oscilloscope's display. For more information, refer to the chapter, "Waveform Commands."

When using GPIB, you may abort a digitize operation by sending a Device Clear over the bus (for example, CLEAR 707).

# Receiving Information from the Oscilloscope

After receiving a query (a command header followed by a question mark), the oscilloscope places the answer in its output queue. The answer remains in the output queue until it is read or until another command is issued. When read, the answer is transmitted across the interface to the computer. The input statement for receiving a response message from an oscilloscope's output queue typically has two parameters; the device address and a format specification for handling the response message. For example, to read the result of the query command :CHANNEL1:INPUT? you would execute the HP BASIC statement:

```
ENTER <device address> ;Setting$
```

This would enter the current setting for the channel 1 coupling in the string variable Setting$. The device address parameter represents the address of the oscilloscope.

All results for queries sent in a program message must be read before another program message is sent. For example, when you send the query :MEASURE:RISETIME?, you must follow that query with an input statement. In HP BASIC, this is usually done with an ENTER statement.

**Handle Queries Properly**

If you send another command or query before reading the result of a query, the output buffer will be cleared and the current response will be lost. This will also generate a query-interrupted error in the error queue. If you execute an input statement before you send a query, it will cause the computer to wait indefinitely.

The format specification for handling response messages depends on both the computer and the programming language.

## String Variable Example

The output of the oscilloscope may be numeric or character data depending on what is queried. Refer to the specific commands for the formats and types of data returned from queries.

For the example programs, assume that the device being programmed is at device address 707. The actual address depends on how you have configured the bus for your own application.

In HP BASIC 5.0, string variables are case-sensitive, and must be expressed exactly the same each time they are used. This example shows the data being returned to a string variable:

```
10 DIM Rang$[30]
20 OUTPUT 707;":CHANNEL1:RANGE?"
30 ENTER 707;Rang$
40 PRINT Rang$
50 END
```

After running this program, the computer displays:

+8.00000E-01

## Numeric Variable Example

This example shows the data being returned to a numeric variable:

```
10 OUTPUT 707;":CHANNEL1:RANGE?"
20 ENTER 707;Rang
30 PRINT Rang
40 END
```

After running this program, the computer displays:

.8

# Definite-Length Block Response Data

Definite-length block response data allows any type of device-dependent data to be transmitted over the system interface as a series of 8-bit binary data bytes. This is particularly useful for sending large quantities of data or 8-bit extended ASCII codes. The syntax is a pound sign ( # ) followed by a non-zero digit representing the number of digits in the decimal integer. After the non-zero digit is the decimal integer that states the number of 8-bit data bytes being sent. This is followed by the actual data.

For example, for transmitting 4000 bytes of data, the syntax would be:

```
#44000 <4000 bytes of data> <terminator>
```

The lifetimes "4" represents the number of digits in the number of bytes, and "4000" represents the number of bytes to be transmitted.

## Multiple Queries

You can send multiple queries to the oscilloscope within a single program message, but you must also read them back within a single program message. This can be accomplished by either reading them back into a string variable or into multiple numeric variables. For example, you could read the result of the query :TIMEBASE:RANGE?;DELAY? into the string variable Results$ with the command:

```
ENTER 707;Results$
```

When you read the result of multiple queries into string variables, each response is separated by a semicolon. For example, the response of the query :TIMEBASE:RANGE?;DELAY? would be:

```
<range_value>;<delay_value>
```

Use the following program message to read the query :TIMEBASE:RANGE?;DELAY? into multiple numeric variables:

```
ENTER 707;Result1,Result2
```

## Oscilloscope Status

Status registers track the current status of the oscilloscope. By checking the oscilloscope status, you can find out whether an operation has completed and is receiving triggers. The chapter, "Status Reporting" explains how to check the status of the oscilloscope.

2

# LAN and GPIB Interfaces

# LAN and GPIB Interfaces

There are two types of interfaces that can be used to remotely program the Infiniium oscilloscope: Local Area Network (LAN) interface and GPIB interface.

## LAN Interface Connector

The oscilloscope is equiped with a LAN interface RJ-45 connector on the rear panel. This allows direct connect to your network. However, before you can use the LAN interface to program the oscilloscope, the network properties must be configured. Unless you are a Network Administrator, you should contact your Network Administrator to add the appropriate client, protocols, and configuration information for your LAN. This information is different for every company.

## GPIB Interface Connector

The oscilloscope is equipped with a GPIB interface connector on the rear panel. This allows direct connection to a GPIB equipped computer. You can connect an external GPIB compatible device to the oscilloscope by installing a GPIB cable between the two units. Finger tighten the captive screws on both ends of the GPIB cable to avoid accidentally disconnecting the cable during operation.

A maximum of fifteen GPIB compatible instruments (including a computer) can be interconnected in a system by stacking connectors. This allows the oscilloscopes to be connected in virtually any configuration, as long as there is a path from the computer to every device operating on the bus.

**CAUTION** Avoid stacking more than three or four cables on any one connector. Multiple connectors produce leverage that can damage a connector mounting.

## Default Startup Conditions

The following default conditions are established during power-up:

- The Request Service (RQS) bit in the status byte register is set to zero.
- All of the event registers are cleared.
- The Standard Event Status Enable Register is set to 0xFF hex.
- Service Request Enable Register is set to 0x80 hex.
- The Operation Status Enable Register is set to 0xFFFF hex.
- The Overload Event Enable Register is set to 0xFF hex.
- The Mask Test Event Enable Register is set to 0xFF hex.

You can change the default conditions using the *PSC command with a parameter of 1 (one). When set to 1, the Standard Event Status Enable Register is set 0x00 hex and the Service Request Enable Register is set to 0x00 hex. This prevents the Power On (PON) event from setting the SRQ interrupt when the oscilloscope is ready to receive commands.

# Interface Capabilities

The interface capabilities of this oscilloscope, as defined by IEEE 488.1 and IEEE 488.2, are listed in Table 2-1.

**Table 2-1 Interface Capabilities**

| Code | Interface Function | Capability |
|---|---|---|
| SH1 | Source Handshake | Full Capability |
| AH1 | Acceptor Handshake | Full Capability |
| T5 | Talker | Basic Talker/Serial Poll/Talk Only Mode/ Unaddress if Listen Address (MLA) |
| L4 | Listener | Basic Listener/ Unaddresses if Talk Address (MTA) |
| SR1 | Service Request | Full Capability |
| RL1 | Remote Local | Complete Capability |
| PP0 | Parallel Poll | No Capability |
| DC1 | Device Clear | Full Capability |
| DT1 | Device Trigger | Full Capability |
| C0 | Computer | No Capability |
| E2 | Driver Electronics | Tri State (1 MB/SEC MAX) |

# GPIB Command and Data Concepts

The GPIB interface has two modes of operation: command mode and data mode. The interface is in the command mode when the Attention (ATN) control line is true. The command mode is used to send talk and listen addresses and various interface commands such as group execute trigger (GET).

The interface is in the data mode when the ATN line is false. The data mode is used to convey device-dependent messages across the bus. The device-dependent messages include all of the oscilloscope-specific commands, queries, and responses found in this manual, including oscilloscope status information.

# Communicating Over the GPIB Interface

Device addresses are sent by the computer in the command mode to specify who talks and who listens. Because GPIB can address multiple devices through the same interface card, the device address passed with the program message must include the correct interface select code and the correct oscilloscope address.

Device Address = (Interface Select Code * 100) + Oscilloscope Address

**The Oscilloscope is at Address 707 for Programming Examples**

The programming examples in this manual assume that the oscilloscope is at device address 707.

### Interface Select Code

Each interface card has a unique interface select code. This code is used by the computer to direct commands and communications to the proper interface. The default is typically "7" for the GPIB interface cards.

### Oscilloscope Address

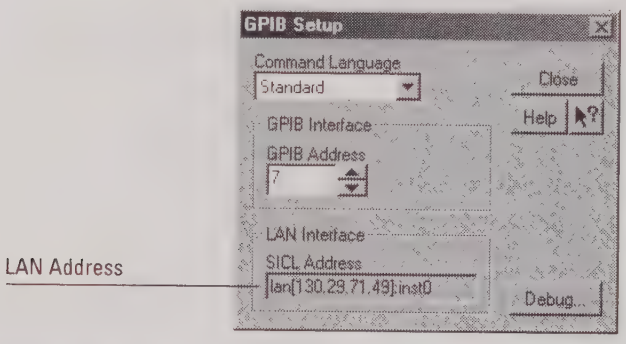
Each oscilloscope on the GPIB must have a unique oscilloscope address between decimal 0 and 30. This oscilloscope address is used by the computer to direct commands and communications to the proper oscilloscope on an interface. The default is typically "7" for this oscilloscope. You can change the oscilloscope address in the Utilities, Remote Interface dialog box.

**Do Not Use Address 21 for an Oscilloscope Address**

Address 21 is usually reserved for the Computer interface Talk/Listen address, and should not be used as an oscilloscope address.

# Communicating Over the LAN Interface

The device address used to send commands and receive data using the LAN interface is located in the GPIB Setup dialog box as shown below.




**GPIB Setup Dialog Box**

The following C example program shows how to communicate with the oscilloscope using the LAN interface and the Agilent Standard Instrument Control Library (SICL).

```
#include <sicl.h>

#define BUFFER_SIZE 100

main()
{
INST Bus;
int reason;
unsigned long actualcnt;
char buffer[ BUFFER_SIZE ];

   /* Open the LAN interface */
   Bus = iopen( "lan[130.29.71.49]:inst0" );
   if( Bus != 0 ) {
      /* Bus timeout set to 20 seconds */
      itimeout( Bus, 20000 );

      /* Clear the interface */
      iclear( Bus );
      /* Query and print the oscilloscope's Id */
      iwrite( Bus, "*IDN?", 5, 1, &actualcnt );
      iread( Bus, buffer, BUFFER_SIZE, &reason, &actualcnt );
      buffer[ actualcnt - 1 ] = 0;

      printf( "%s\n", buffer );
      iclose( Bus );
   }
}
```

# Bus Commands

The following commands are IEEE 488.1 bus commands (ATN true). IEEE 488.2 defines many of the actions that are taken when these commands are received by the oscilloscope.

### Device Clear

The device clear (DCL) and selected device clear (SDC) commands clear the input buffer and output queue, reset the parser, and clear any pending commands. If either of these commands is sent during a digitize operation, the digitize operation is aborted.

### Group Execute Trigger

The group execute trigger (GET) command arms the trigger. This is the same action produced by sending the RUN command.

### Interface Clear

The interface clear (IFC) command halts all bus activity. This includes unaddressing all listeners and the talker, disabling serial poll on all devices, and returning control to the system computer.

3

# Message Communication and System Functions

# Message Communication and System Functions

This chapter describes the operation of oscilloscopes that operate in compliance with the IEEE 488.2 (syntax) standard. It is intended to give you enough basic information about the IEEE 488.2 standard to successfully program the oscilloscope. You can find additional detailed information about the IEEE 488.2 standard in ANSI/IEEE Std 488.2-1987, *"IEEE Standard Codes, Formats, Protocols, and Common Commands."*

This oscilloscope series is designed to be compatible with other Agilent Technologies IEEE 488.2 compatible instruments. Oscilloscopes that are compatible with IEEE 488.2 must also be compatible with IEEE 488.1 (GPIB bus standard); however, IEEE 488.1 compatible oscilloscopes may or may not conform to the IEEE 488.2 standard. The IEEE 488.2 standard defines the message exchange protocols by which the oscilloscope and the computer will communicate. It also defines some common capabilities that are found in all IEEE 488.2 oscilloscopes. This chapter also contains some information about the message communication and system functions not specifically defined by IEEE 488.2.

# Protocols

The message exchange protocols of IEEE 488.2 define the overall scheme used by the computer and the oscilloscope to communicate. This includes defining when it is appropriate for devices to talk or listen, and what happens when the protocol is not followed.

### Functional Elements

Before proceeding with the description of the protocol, you should understand a few system components, as described here.

**Input Buffer** — The input buffer of the oscilloscope is the memory area where commands and queries are stored prior to being parsed and executed. It allows a computer to send a string of commands, which could take some time to execute, to the oscilloscope, then proceed to talk to another oscilloscope while the first oscilloscope is parsing and executing commands.

**Output Queue** — The output queue of the oscilloscope is the memory area where all output data or response messages are stored until read by the computer.

**Parser** — The oscilloscope's parser is the component that interprets the commands sent to the oscilloscope and decides what actions should be taken. "Parsing" refers to the action taken by the parser to achieve this goal. Parsing and execution of commands begins when either the oscilloscope recognizes a program message terminator, or the input buffer becomes full. If you want to send a long sequence of commands to be executed, then talk to another oscilloscope while they are executing, you should send all of the commands before sending the program message terminator.

### Protocol Overview

The oscilloscope and computer communicate using program messages and response messages. These messages serve as the containers into which sets of program commands or oscilloscope responses are placed.

A program message is sent by the computer to the oscilloscope, and a response message is sent from the oscilloscope to the computer in response to a query message. A query message is defined as being a program message that contains one or more queries. The oscilloscope will only talk when it has received a valid query message, and therefore has something to say. The computer should only attempt to read a response after sending a complete query message, but before sending another program message.

**Remember this Rule of Oscilloscope Communication**

The basic rule to remember is that the oscilloscope will only talk when prompted to, and it then expects to talk before being told to do something else.

### Protocol Operation

When you turn the oscilloscope on, the input buffer and output queue are cleared, and the parser is reset to the root level of the command tree.

The oscilloscope and the computer communicate by exchanging complete program messages and response messages. This means that the computer should always terminate a program message before attempting to read a response. The oscilloscope will terminate response messages except during a hard copy output.

After you send a query message, the next message should be the response message. The computer should always read the complete response message associated with a query message before sending another program message to the same oscilloscope.

The oscilloscope allows the computer to send multiple queries in one query message. This is called sending a "compound query." Multiple queries in a query message are separated by semicolons. The responses to each of the queries in a compound query will also be separated by semicolons.

Commands are executed in the order they are received.

### Protocol Exceptions

If an error occurs during the information exchange, the exchange may not be completed in a normal manner.

### Suffix Multiplier

The suffix multipliers that the oscilloscope will accept are shown in Table 3-1.

**Table 3-1** **<suffix mult>**

| Value | Mnemonic | Value | Mnemonic |
|---|---|---|---|
| 1E18 | EX | 1E-3 | M |
| 1E15 | PE | 1E-6 | U |
| 1E12 | T | 1E-9 | N |
| 1E9 | G | 1E-12 | P |
| 1E6 | MA | 1E-15 | F |
| 1E3 | K | 1E-18 | A |

### Suffix Unit

The suffix units that the oscilloscope will accept are shown in Table 3-2.

**Table 3-2** **<suffix unit>**

| Suffix | Referenced Unit |
|---|---|
| V | Volt |
| S | Second |

4

# Status Reporting

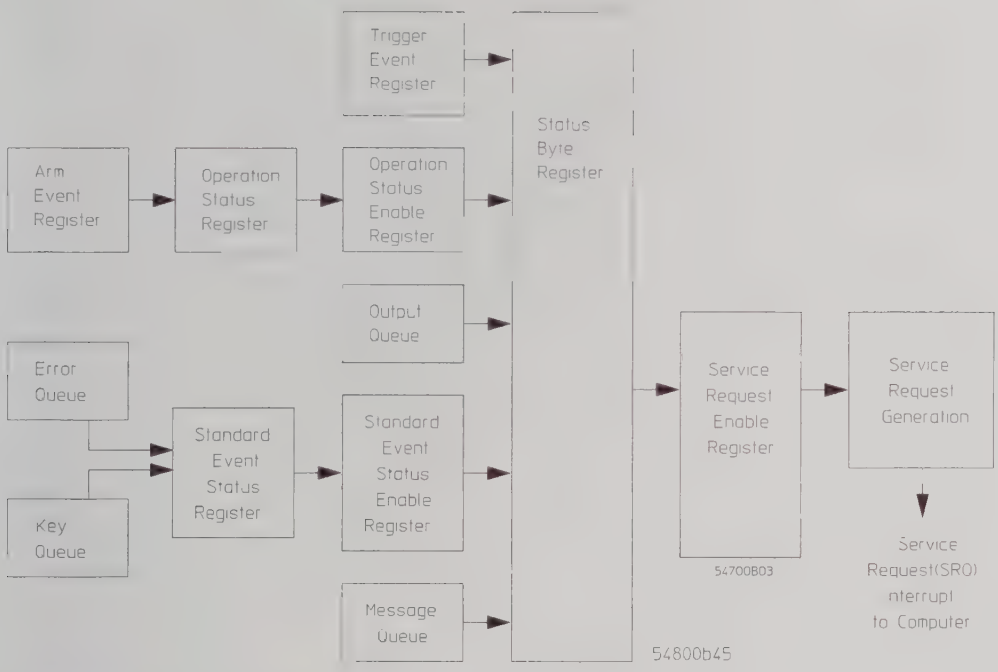
# Status Reporting

An overview of the oscilloscope's status reporting structure is shown in Figure 4-1. The status reporting structure shows you how to monitor specific events in the oscilloscope. Monitoring these events lets you determine the status of an operation, the availability and reliability of the measured data, and more.

- To monitor an event, first clear the event, then enable the event. All of the events are cleared when you initialize the oscilloscope.
- To generate a service request (SRQ) interrupt to an external computer, enable at least one bit in the Status Byte Register.

The Status Byte Register, the Standard Event Status Register group, and the Output Queue are defined as the Standard Status Data Structure Model in IEEE 488.2-1987. IEEE 488.2 defines data structures, commands, and common bit definitions for status reporting. There are also oscilloscope-defined structures and bits.

**Figure 4-1**




**Status Reporting Overview Block Diagram**

The status reporting structure consists of the registers shown here.

Table 4-1 lists the bit definitions for each bit in the status reporting data structure.

**Table 4-1** **Status Reporting Bit Definition**

| Bit | Description | Definition |
|---|---|---|
| PON | Power On | Indicates power is turned on. |
| URQ | User Request | Not Used. Permanently set to zero. |
| CME | Command Error | Indicates if the parser detected an error. |
| EXE | Execution Error | Indicates if a parameter was out of range or was inconsistent with the current settings. |

| Bit | Description | Definition |
|---|---|---|
| DDE | Device Dependent Error | Indicates if the device was unable to complete an operation for device-dependent reasons. |
| QYE | Query Error | Indicates if the protocol for queries has been violated. |
| RQL | Request Control | Indicates if the device is requesting control. |
| OPC | Operation Complete | Indicates if the device has completed all pending operations. |
| OPER | Operation Status Register | Indicates if any of the enabled conditions in the Operation Status Register have occurred. |
| RQS | Request Service | Indicates that the device is requesting service. |
| MSS | Master Summary Status | Indicates if a device has a reason for requesting service. |
| ESB | Event Status Bit | Indicates if any of the enabled conditions in the Standard Event Status Register have occurred. |
| MAV | Message Available | Indicates if there is a response in the output queue. |
| MSG | Message | Indicates if an advisory has been displayed. |
| USR | User Event Register | Indicates if any of the enabled conditions have occurred in the User Event Register. |
| TRG | Trigger | Indicates if a trigger has been received. |
| WAIT TRIG | Wait for Trigger | Indicates the oscilloscope is armed and ready for trigger. |

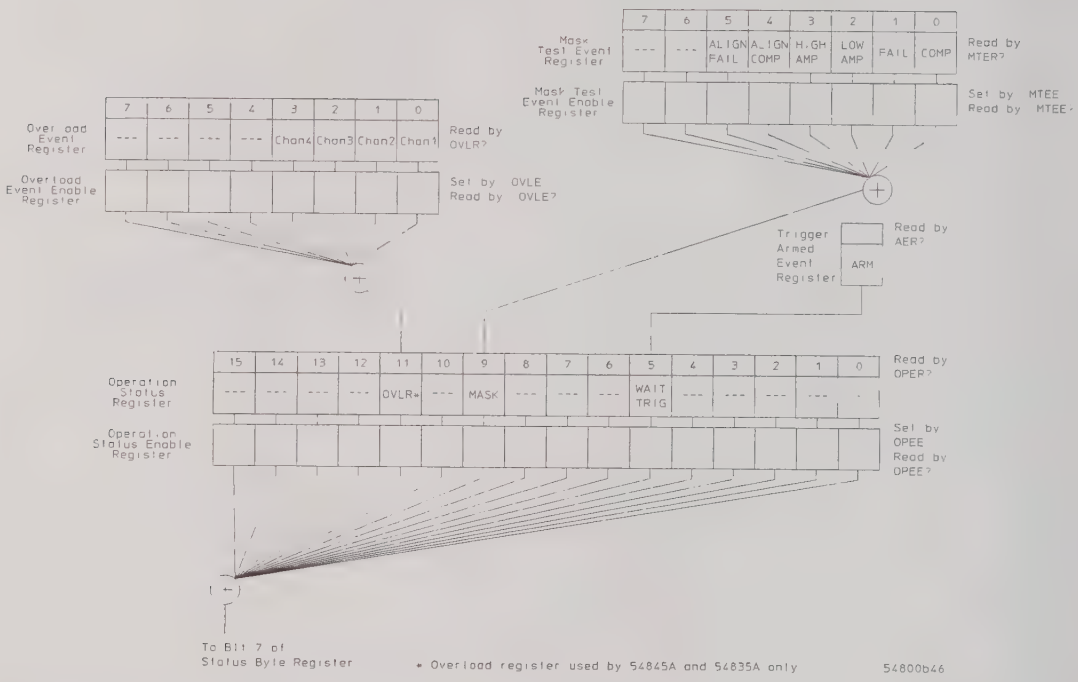
## Status Reporting Data Structures

The different status reporting data structures, descriptions, and interactions are shown in Figure 4-2. To make it possible for any of the Standard Event Status Register bits to generate a summary bit, you must enable the corresponding bits. These bits are enabled by using the *ESE common command to set the corresponding bit in the Standard Event Status Enable Register.

To generate a service request (SRQ) interrupt to the computer, you must enable at least one bit in the Status Byte Register. These bits are enabled by using the *SRE common command to set the corresponding bit in the Service Request Enable Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

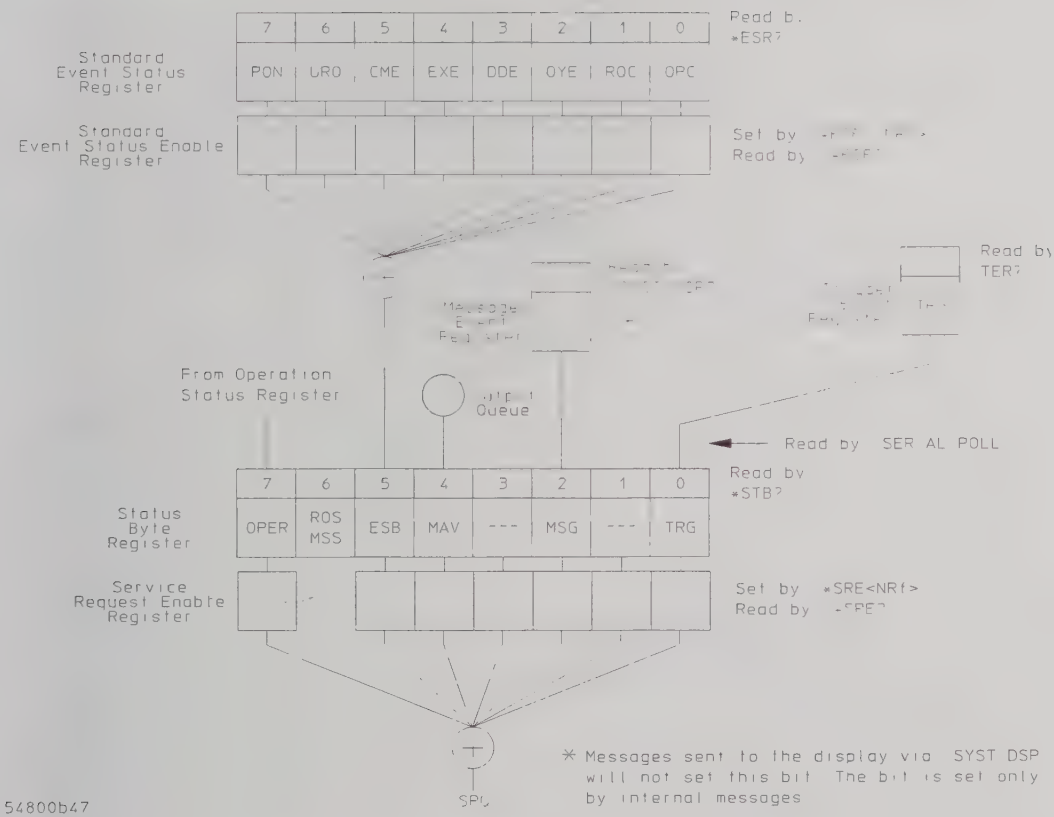
For more information about common commands, see the "Common Commands" chapter.

**Figure 4-2**


7 6 5 4 3 2 1 0
Mask Test Event Register
--- --- ALIGN FAIL ALIGN COMP HIGH AMP LOW AMP FAIL COMP
Read by MTER?
Mask Test Event Enable Register
Set by MTEE
Read by MTEE?
Overload Event Register
--- --- --- --- Chan4 Chan3 Chan2 Chan1
Read by OVLR?
Overload Event Enable Register
Set by OVLE
Read by OVLE?
Trigger Armed Event Register
ARM
Read by AER?
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Operation Status Register
--- --- --- --- OVLR* --- MASK --- --- --- WAIT TRIG --- --- --- --- ---
Read by OPER?
Operation Status Enable Register
Set by OPEE
Read by OPEE?
To Bit 7 of Status Byte Register
* Overload register used by 54845A and 54835A only
54800b46

**Status Reporting Data Structures**

**Figure 4-2 (Continued)**


7 6 5 4 3 2 1 0
Standard Event Status Register
PON URQ CME EXE DDE QYE ROC OPC
Standard Event Status Enable Register
Read by *ESR?
Read by TER?
From Operation Status Register
Output Queue
Read by SER AL POLL
Read by *STB?
Status Byte Register
OPER ROS MSS ESB MAV --- MSG --- TRG
Service Request Enable Register
Set by *SRE<NRf>
Read by *SRE?
SPU
* Messages sent to the display via SYST DSP will not set this bit. The bit is set only by internal messages
54800b47

**Status Reporting Data Structures (Continued)**

## Status Byte Register

The Status Byte Register is the summary-level register in the status reporting structure. It contains summary bits that monitor activity in the other status registers and queues. The Status Byte Register is a live register. That is, its summary bits are set and cleared by the presence and absence of a summary bit from other event registers or queues.

If the Status Byte Register is to be used with the Service Request Enable Register to set bit 6 (RQS/MSS) and to generate an SRQ, at least one of the summary bits must be enabled, then set. Also, event bits in all other status registers must be specifically enabled to generate the summary bit that sets the associated summary bit in the Status Byte Register.

You can read the Status Byte Register using either the *STB? common command query or the GPIB serial poll command. Both commands return the decimal-weighted sum of all set bits in the register. The difference between the two methods is that the serial poll command reads bit 6 as the Request Service (RQS) bit and clears the bit which clears the SRQ interrupt. The *STB? query reads bit 6 as the Master Summary Status (MSS) and does not clear the bit or have any effect on the SRQ interrupt. The value returned is the total bit weights of all of the bits that are set at the present time.

The use of bit 6 can be confusing. This bit was defined to cover all possible computer interfaces, including a computer that could not do a serial poll. The important point to remember is that if you are using an SRQ interrupt to an external computer, the serial poll command clears bit 6. Clearing bit 6 allows the oscilloscope to generate another SRQ interrupt when another enabled event occurs.

The only other bit in the Status Byte Register affected by the *STB? query is the Message Available bit (bit 4). If there are no other messages in the Output Queue, bit 4 (MAV) can be cleared as a result of reading the response to the *STB? query.

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, a program would print the sum of the two weights. Since these bits were not enabled to generate an SRQ, bit 6 (weight = 64) is not set.

**Example 1**

This HP BASIC example uses the *STB? query to read the contents of the oscilloscope's Status Byte Register when none of the register's summary bits are enabled to generate an SRQ interrupt.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF;*STB?"     !Turn headers off
20  ENTER 707;Result      !Place result in a numeric variable
30  PRINT Result      !Print the result
40  End
```

The next program prints 132 and clears bit 6 (RQS) of the Status Byte Register. The difference in the decimal value between this example and the previous one is the value of bit 6 (weight = 64). Bit 6 is set when the first enabled summary bit is set, and is cleared when the Status Byte Register is read by the serial poll command.

**Example 2**

This example uses the HP BASIC serial poll (SPOLL) command to read the contents of the oscilloscope's Status Byte Register.

```
10  Result = SPOLL(707)
20  PRINT Result
30  END
```

**Use Serial Polling to Read the Status Byte Register**

Serial polling is the preferred method to read the contents of the Status Byte Register because it resets bit 6 and allows the next enabled event that occurs to generate a new SRQ interrupt.

## Service Request Enable Register

Setting the Service Request Enable Register bits enables corresponding bits in the Status Byte Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

Bits are set in the Service Request Enable Register using the *SRE command, and the bits that are set are read with the *SRE? query. Bit 6 always returns 0. Refer to the Status Reporting Data Structures shown in Figure 4-2.

**Example**

This example sets bit 4 (MAV) and bit 5 (ESB) in the Service Request Enable Register.

```
OUTPUT 707;"*SRE 48"
```

This example uses the parameter "48" to allow the oscilloscope to generate an SRQ interrupt under the following conditions:

- When one or more bytes in the Output Queue set bit 4 (MAV).
- When an enabled event in the Standard Event Status Register generates a summary bit that sets bit 5 (ESB).

## Message Event Register

This register sets the MSG bit in the status byte register when an internally generated message is written to the advisory line on the oscilloscope. The message is read using the :SYSTEM:DSP? query. Note that messages written to the advisory line on the oscilloscope using the :SYSTEM:DSP command does not set the MSG status bit.

## Trigger Event Register

This register sets the TRG bit in the status byte register when a trigger event occurs.

The trigger event register stays set until it is cleared by reading the register with the TER? query or by using the *CLS (clear status) command. If your application needs to detect multiple triggers, the trigger event register must be cleared after each one.

If you are using the Service Request to interrupt a computer operation when the trigger bit is set, you must clear the event register after each time it is set.

# Standard Event Status Register

The Standard Event Status Register (SESR) monitors the following oscilloscope status events:

- PON - Power On
- CME - Command Error
- EXE - Execution Error
- DDE - Device Dependent Error
- QYE - Query Error
- RQC - Request Control
- OPC - Operation Complete

When one of these events occurs, the corresponding bit is set in the register. If the corresponding bit is also enabled in the Standard Event Status Enable Register, a summary bit (ESB) in the Status Byte Register is set.

You can read the contents of the Standard Event Status Register and clear the register by sending the *ESR? query. The value returned is the total bit weights of all bits set at the present time.

**Example**

This example uses the *ESR? query to read the contents of the Standard Event Status Register.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"      !Turn headers off
20  OUTPUT 707;"*ESR?"
30  ENTER 707;Result      !Place result in a numeric variable
40  PRINT Result      !Print the result
50  End
```

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, the program prints the sum of the two weights.

# Standard Event Status Enable Register

For any of the Standard Event Status Register bits to generate a summary bit, you must first enable the bit. Use the *ESE (Event Status Enable) common command to set the corresponding bit in the Standard Event Status Enable Register. Set bits are read with the *ESE? query.

**Example**

Suppose your application requires an interrupt whenever any type of error occurs. The error status bits in the Standard Event Status Register are bits 2 through 5. The sum of the decimal weights of these bits is 60. Therefore, you can enable any of these bits to generate the summary bit by sending:

```
OUTPUT 707;"*ESE 60"
```

Whenever an error occurs, the oscilloscope sets one of these bits in the Standard Event Status Register. Because the bits are all enabled, a summary bit is generated to set bit 5 (ESB) in the Status Byte Register.

If bit 5 (ESB) in the Status Byte Register is enabled (via the *SRE command), a service request interrupt (SRQ) is sent to the external computer.

**Disabled Standard Event Status Register Bits Respond, but Do Not Generate a Summary Bit**

Standard Event Status Register bits that are not enabled still respond to their corresponding conditions (that is, they are set if the corresponding event occurs). However, because they are not enabled, they do not generate a summary bit in the Status Byte Register.

## Operation Status Register

This register hosts the following bits:

- WAIT TRIG bit (bit 5)
- Mask Test Summary bit (bit 9)
- Overload Summary bit (bit 11)

The WAIT TRIG bit is set by the Trigger Armed Event Register and indicates the trigger is armed.

The Mask Test Summary bit is set whenever at least one of the Mask Test Event Register bits is enabled.

The Overload Summary bit is set whenever at least one of the Overload Event Register bits is enabled (this register is only used by the 54845A and the 54835A).

If any of these bits are set, the OPER bit (bit 7) of the Status Byte Register is set. The Operation Status Register is read and cleared with the OPER? query. The register output is enabled or disabled using the mask value supplied with the OPEE command.

# Operation Status Enable Register

For any of the Operation Status Register bits to generate a summary bit, you must first enable the bit. Use the OPEE (Operation Event Status Enable) command to set the corresponding bit in the Operation Status Enable Register. Set bits are read with the OPEE? query.

**Example**

Suppose your application requires an interrupt whenever any event occurs in the mask test register. The error status bit in the Operation Status Register is bit 9. Therefore, you can enable this bit to generate the summary bit by sending:

```
OUTPUT 707;"OPEE 512" ( hex 200 )
```

Whenever an error occurs, the oscilloscope sets this bit in the Mask Test Event Register. Because this bit is enabled, a summary bit is generated to set bit 9 (OPER) in the Operation Status Register.

If bit 7 (OPER) in the Status Byte Register is enabled (via the *SRE command), a service request interrupt (SRQ) is sent to the external computer.

**Disabled Operation Status Register Bits Respond, but Do Not Generate a Summary Bit**

Operation Status Register bits that are not enabled still respond to their corresponding conditions (that is, they are set if the corresponding event occurs). However, because they are not enabled, they do not generate a summary bit in the Status Byte Register.

## Mask Test Event Register

This register hosts the following bits:

- Mask Test Complete bit (bit 0)
- Mask Test Fail bit (bit 1)
- Mask Low Amplitude bit (bit 2)
- Mask High Amplitude bit (bit 3)
- Mask Align Complete bit (bit 4)
- Mask Align Fail bit (bit 5)

The Mask Test Complete bit is set whenever the mask test is complete.

The Mask Test Fail bit is set whenever the mask test failed.

The Mask Low Amplitude bit is set whenever the signal is below the mask amplitude.

The Mask High Amplitude bit is set whenever the signal is above the mask amplitude.

The Mask Align Complete bit is set whenever the mask align is complete.

The Mask Align Fail bit is set whenever the mask align failed.

If any of these bits are set, the MASK bit (bit 9) of the Operation Status Register is set. The Mask Test Event Register is read and cleared with the MTER? query. The register output is enabled or disabled using the mask value supplied with the MTEE command.

# Mask Test Event Enable Register

For any of the Mask Test Event Register bits to generate a summary bit, you must first enable the bit. Use the MTEE (Mask Test Event Enable) command to set the corresponding bit in the Mask Test Event Enable Register. Set bits are read with the MTEE? query.

**Example**

Suppose your application requires an interrupt whenever a Mask Test Fail occurs in the mask test register. You can enable this bit to generate the summary bit by sending:

```
OUTPUT 707;"MTEE 2"
```

Whenever an error occurs, the oscilloscope sets the MASK bit in the Operation Status Register. Because the bits in the Operation Status Enable Register are all enabled, a summary bit is generated to set bit 7 (OPER) in the Status Byte Register.

If bit 7 (OPER) in the Status Byte Register is enabled (via the *SRE command), a service request interrupt (SRQ) is sent to the external computer.

**Disabled Mask Test Event Register Bits Respond, but Do Not Generate a Summary Bit**

Mask Test Event Register bits that are not enabled still respond to their corresponding conditions (that is, they are set if the corresponding event occurs). However, because they are not enabled, they do not generate a summary bit in the Operation Status Register.

## Trigger Armed Event Register

This register sets bit 5 (Wait Trig bit) in the Operation Status Register and bit 7 (OPER bit) in the Status Byte Register when the oscilloscope becomes armed.

The ARM event register stays set until it is cleared by reading the register with the AER? query or by using the *CLS command. If your application needs to detect multiple triggers, the ARM event register must be cleared after each one.

If you are using the Service Request to interrupt the computer operation when the trigger bit is set, you must clear the event register after each time it is set.

## Error Queue

As errors are detected, they are placed in an error queue. This queue is a first-in, first-out queue. If the error queue overflows, the last error in the queue is replaced with error -350, "Queue overflow." Any time the queue overflows, the oldest errors remain in the queue, and the most recent error is discarded. The length of the oscilloscope's error queue is 30 (29 positions for the error messages, and 1 position for the "Queue overflow" message).

The error queue is read with the :SYSTEM:ERROR? query. Executing this query reads and removes the oldest error from the head of the queue, which opens a position at the tail of the queue for a new error. When all the errors have been read from the queue, subsequent error queries return 0, "No error."

The error queue is cleared when any of these events occur:

- When the oscilloscope is powered up.
- When the oscilloscope receives the *CLS common command.
- When the last item is read from the error queue.

For more information on reading the error queue, refer to the :SYSTEM:ERROR? query in the System Commands chapter. For a complete list of error messages, refer to the chapter, "Error Messages."

## Output Queue

The output queue stores the oscilloscope-to-computer responses that are generated by certain oscilloscope commands and queries. The output queue generates the Message Available summary bit when the output queue contains one or more bytes. This summary bit sets the MAV bit (bit 4) in the Status Byte Register. You may read the output queue with the HP Basic ENTER statement.

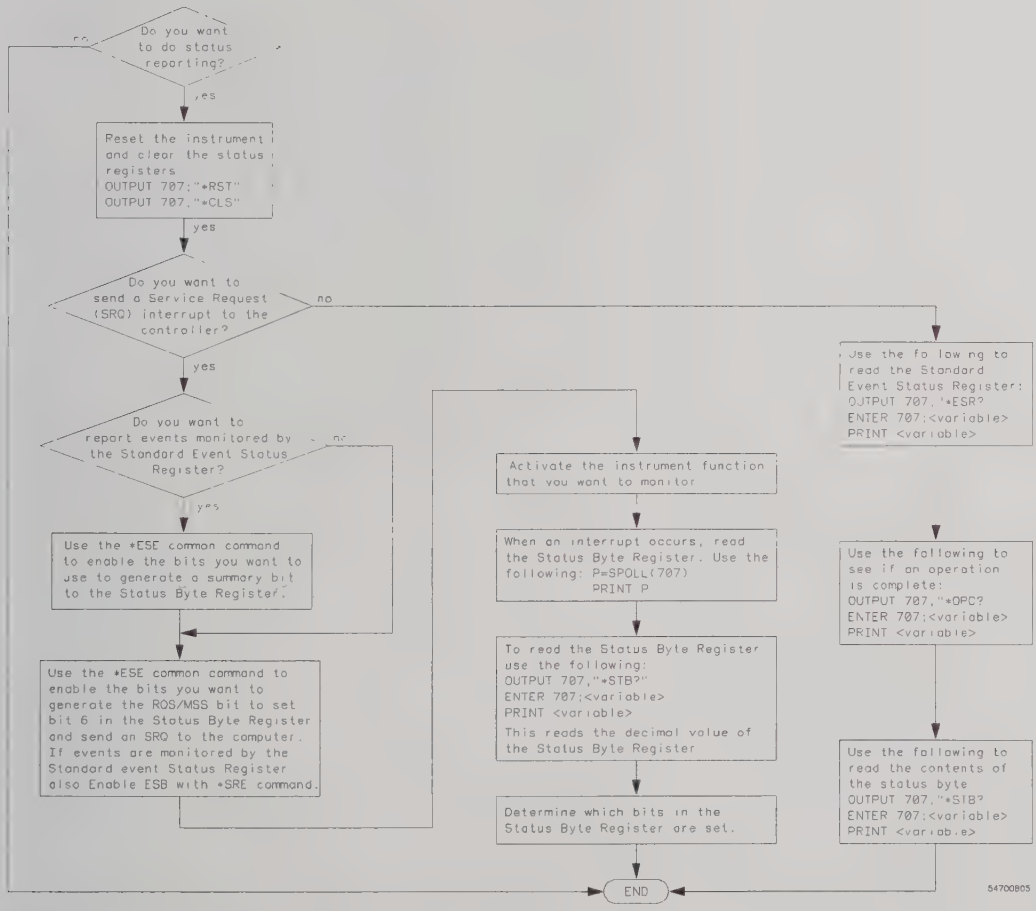
## Message Queue

The message queue contains the text of the last message written to the advisory line on the screen of the oscilloscope. The queue is read with the :SYSTEM:DSP? query. Note that messages sent with the :SYSTEM:DSP command do not set the MSG status bit in the Status Byte Register.

## Clearing Registers and Queues

The *CLS common command clears all event registers and all queues except the output queue. If *CLS is sent immediately following a program message terminator, the output queue is also cleared.

**Figure 4-3**


Do you want to do status reporting?
no
yes
Reset the instrument and clear the status registers
OUTPUT 707;"*RST"
OUTPUT 707;"*CLS"
yes
Do you want to send a Service Request (SRQ) interrupt to the controller?
no
yes
Use the following to read the Standard Event Status Register:
OUTPUT 707;"*ESR?
ENTER 707;<variable>
PRINT <variable>
Do you want to report events monitored by the Standard Event Status Register?
no
yes
Activate the instrument function that you want to monitor
When an interrupt occurs, read the Status Byte Register. Use the following: P=SPOLL(707)
PRINT P
Use the *ESE common command to enable the bits you want to use to generate a summary bit to the Status Byte Register.
Use the following to see if an operation is complete:
OUTPUT 707;"*OPC?
ENTER 707;<variable>
PRINT <variable>
Use the *ESE common command to enable the bits you want to generate the ROS/MSS bit to set bit 6 in the Status Byte Register and send an SRQ to the computer. If events are monitored by the Standard event Status Register also Enable ESB with *SRE command.
To read the Status Byte Register use the following:
OUTPUT 707;"*STB?"
ENTER 707;<variable>
PRINT <variable>
This reads the decimal value of the Status Byte Register
Use the following to read the contents of the status byte
OUTPUT 707;"*STB?
ENTER 707;<variable>
PRINT <variable>
Determine which bits in the Status Byte Register are set.
END
54700B05

**Status Reporting Decision Chart**

5

# Programming Conventions

# Programming Conventions

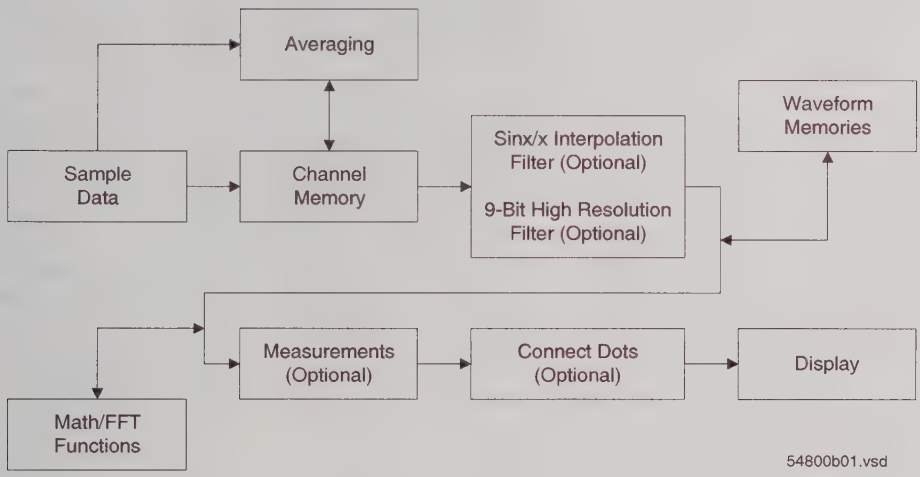
This chapter describes conventions used to program the Infiniium-Series Oscilloscopes, and conventions used throughout this manual. A block diagram and description of data flow is included for understanding oscilloscope operations. A description of the command tree and command tree traversal is also included. Also see the Programmer's Quick Reference Guide for more information about command syntax.

# Data Flow

The data flow gives you an idea of where the measurements are made on the acquired data, and when the post-waveform processing is applied to the data.

Figure 5-1 is a block diagram of the oscilloscope. The diagram is laid out serially for a visual perception of how the data is affected by the oscilloscope.

**Figure 5-1**




**Sample Data Processing**

The sample data is stored in the channel memory for further processing before being displayed. The time it takes for the sample data to be displayed depends on the number of post processes you have selected.

Averaging your sampled data helps remove any unwanted noise from your waveform. The 9-bit, high-resolution filter also removes noise from your waveform by limiting the bandwidth of the oscilloscope to:

$F_s/20$

where:

$F_s$ = the sampling frequency

This filter lowers the noise floor of the oscilloscope, which increases the oscilloscope's vertical resolution.

You can store your sample data in Infiniium's waveform memories for use as one of the sources in Math/FFT functions, or to visually compare against a waveform that is captured at a future time. The Math/FFT functions let you apply mathematical operations on your sampled data. You can use these functions to duplicate many of the mathematical operations that your circuit may be performing to verify that your circuit is operating correctly.

The measurements section performs any of the automated measurements that are available in Infiniium. The measurements that you have selected appear at the bottom of the display.

The Connect Dots section draws a straight line between sample data points, giving an analog look to the waveform. This is sometimes called linear interpolation.

## Truncation Rule

The truncation rule is used to produce the short form (abbreviated spelling) for the mnemonics used in the programming headers and alpha arguments.

**Command Truncation Rule**

The mnemonic is the first four characters of the keyword, unless the fourth character is a vowel. Then the mnemonic is the first three characters of the keyword. If the length of the keyword is four characters or less, this rule does not apply, and the short form is the same as the long form.

Table 5-1 shows how the truncation rule is applied to commands.

**Table 5-1 Mnemonic Truncation**

| Long Form | Short Form | How the Rule is Applied |
|---|---|---|
| RANGE | RANG | Short form is the first four characters of the keyword. |
| PATTERN | PATT | Short form is the first four characters of the keyword. |
| DISK | DISK | Short form is the same as the long form. |
| DELAY | DEL | Fourth character is a vowel; short form is the first three characters. |

# The Command Tree

The command tree in Figure 5-2 shows all of the commands in the Infiniium-Series Oscilloscopes and the relationship of the commands to each other. The IEEE 488.2 common commands are not listed as part of the command tree because they do not affect the position of the parser within the tree.

When a program message terminator (<NL>, linefeed - ASCII decimal 10) or a leading colon (:) is sent to the oscilloscope, the parser is set to the "root" of the command tree.

### Command Types

The commands in this oscilloscope can be viewed as three types: common commands, root level commands, and subsystem commands.

- Common commands are commands defined by IEEE 488.2 and control some functions that are common to all IEEE 488.2 instruments. These commands are independent of the tree and do not affect the position of the parser within the tree. *RST is an example of a common command.
- Root level commands control many of the basic functions of the oscilloscope. These commands reside at the root of the command tree. They can always be parsed if they occur at the beginning of a program message or are preceded by a colon. Unlike common commands, root level commands place the parser back at the root of the command tree. AUTOSCALE is an example of a root level command.
- Subsystem commands are grouped together under a common node of the command tree, such as the TIMEBASE commands. You may select only one subsystem at a given time. When you turn on the oscilloscope initially, the command parser is set to the root of the command tree and no subsystem is selected.

**See Also**

The Programmer's Quick Reference Guide for information about command syntax and command syntax diagrams.

### Tree Traversal Rules

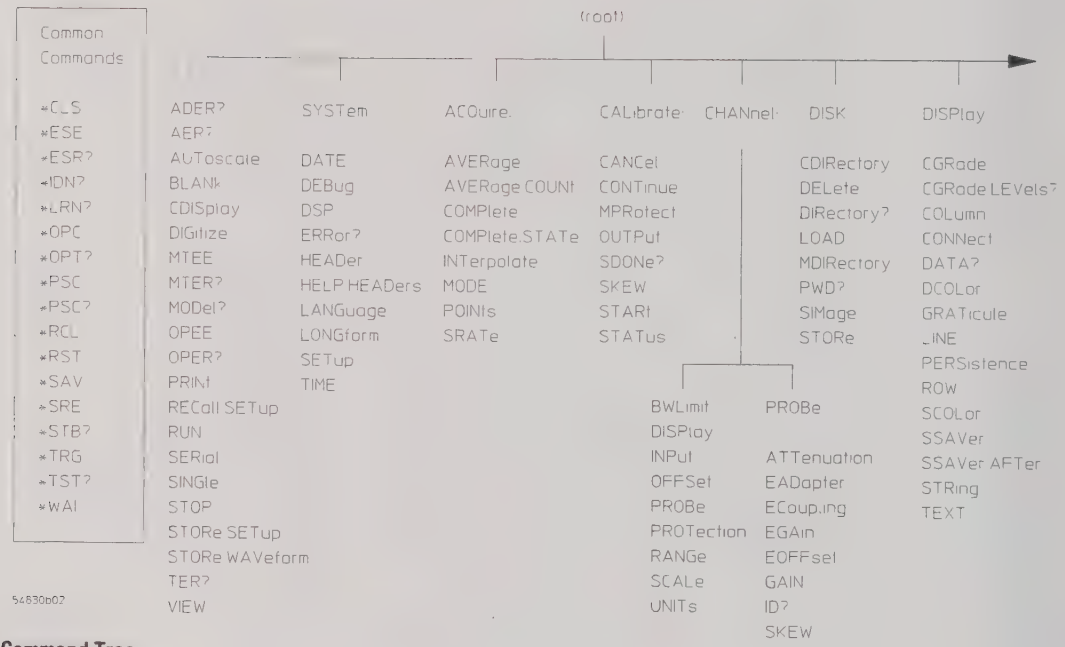
Command headers are created by traversing down the command tree. A legal command header from the command tree would be :TIMEBASE:RANGE. This is referred to as a compound header. A compound header is a header made up of two or more mnemonics separated by colons. The compound header contains no spaces. The following rules apply to traversing the tree.

**Tree Traversal Rules**

A leading colon or a program message terminator (<NL> or EOI true on the last byte) places the parser at the root of the command tree. A leading colon is a colon that is the first character of a program header. Executing a subsystem command places the oscilloscope in that subsystem until a leading colon or a program message terminator is found.

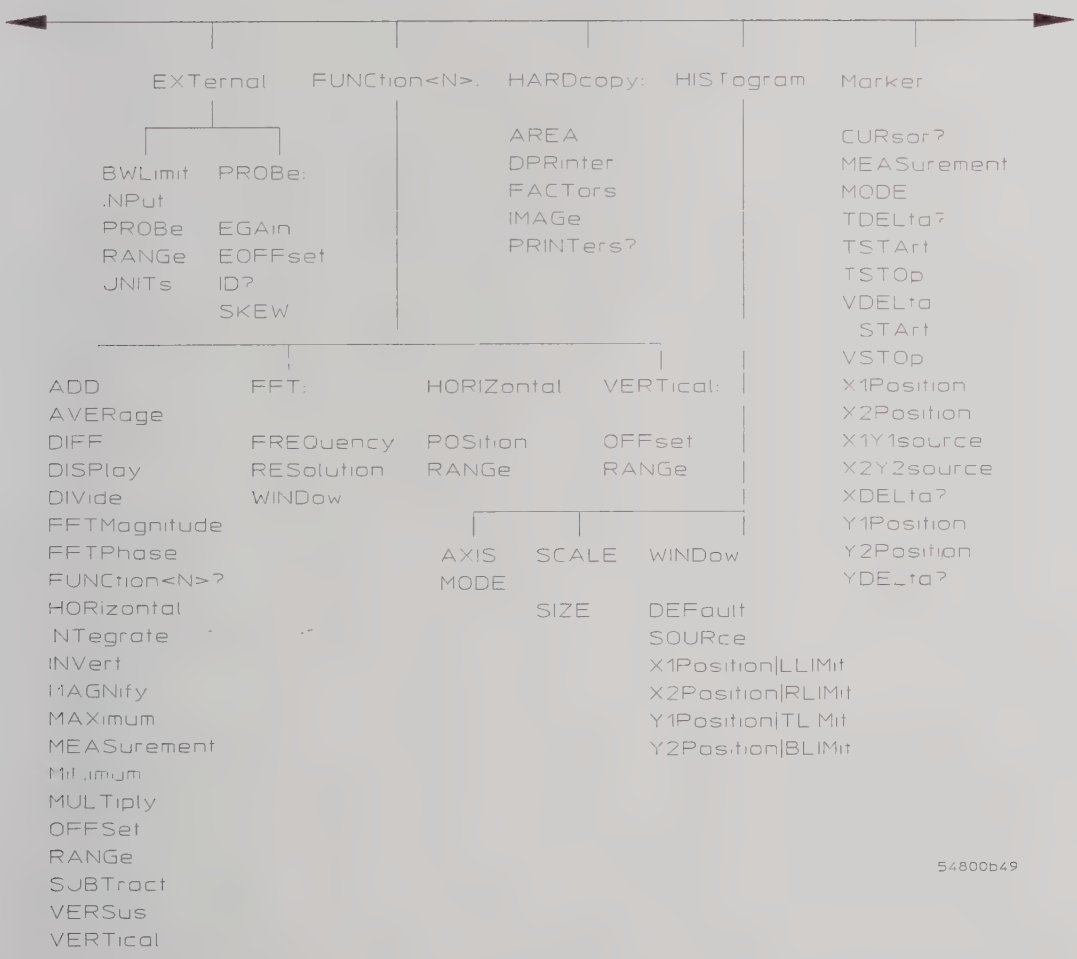
In the command tree, use the last mnemonic in the compound header as a reference point (for example, RANGE). Then find the last colon above that mnemonic (TIMEBASE:). That is the point where the parser resides. You can send any command below this point within the current program message without sending the mnemonics which appear above them (for example, REFERENCE).

**Figure 5-2**



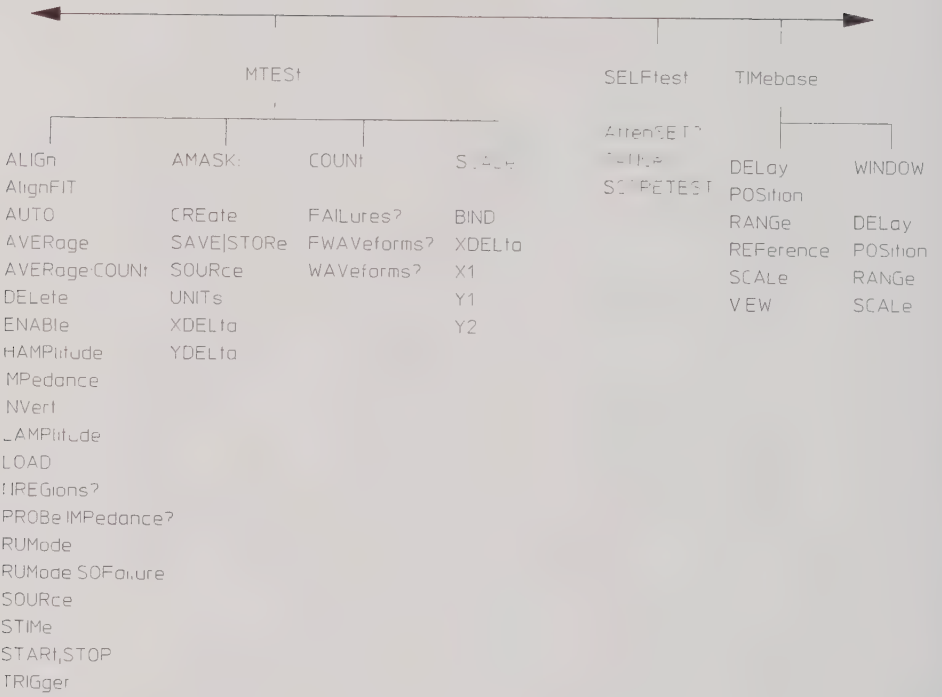

**Command Tree**

**Figure 5-3**



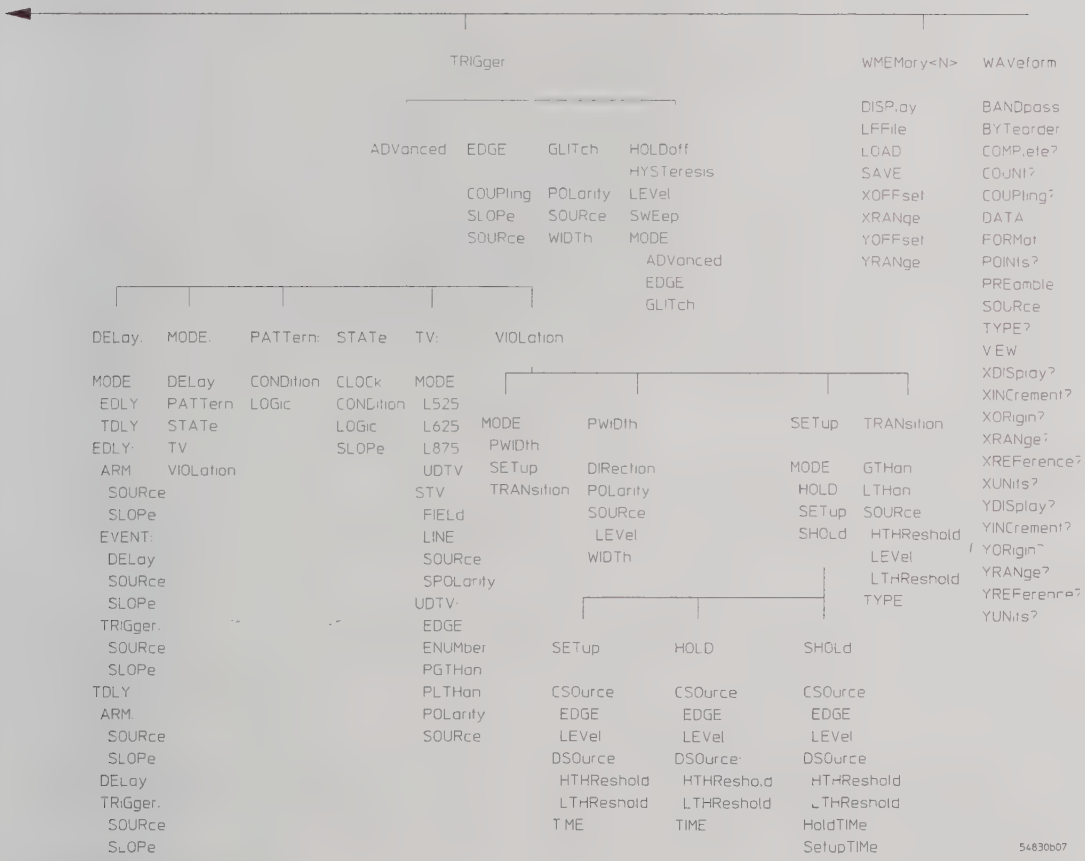

**Command Tree (Continued)**

**Figure 5-4**




**Command Tree (Continued)**

**Figure 5-5**




**Command Tree (Continued)**

### Tree Traversal Examples

The OUTPUT statements in the following examples are written using HP BASIC 5.0. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

**Example 1**

Consider the following command:

```
OUTPUT 707;":CHANNEL1:RANGE 0.5;OFFSET 0"
```

The colon between CHANNEL1 and RANGE is necessary because :CHANNEL1:RANGE is a compound command. The semicolon between the RANGE command and the OFFSET command is required to separate the two commands or operations. The OFFSET command does not need :CHANNEL1 preceding it because the :CHANNEL1:RANGE command sets the parser to the CHANNEL1 node in the tree.

**Example 2**

Consider the following commands:

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER;POSITION 0.00001"
```

or

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER"
OUTPUT 707;":TIMEBASE:POSITION 0.00001"
```

In the first line of example 2, the "subsystem selector" is implied for the POSITION command in the compound command.

A second way to send these commands is shown in the second part of the example. Because the program message terminator places the parser back at the root of the command tree, you must reselect TIMEBASE to re-enter the TIMEBASE node before sending the POSITION command.

**Example 3**

Consider the following command:

```
OUTPUT 707;":TIMEBASE:REFERENCE CENTER;:CHANNEL1:OFFSET 0"
```

In this example, the leading colon before CHANNEL1 tells the parser to go back to the root of the command tree. The parser can then recognize the :CHANNEL1:OFFSET command and enter the correct node.

## Infinity Representation

The representation for infinity for this oscilloscope is 9.99999E+37. This is also the value returned when a measurement cannot be made.

## Sequential and Overlapped Commands

IEEE 488.2 makes a distinction between sequential and overlapped commands. Sequential commands finish their task before the execution of the next command starts. Overlapped commands run concurrently. Commands following an overlapped command may be started before the overlapped command is completed.

## Response Generation

As defined by IEEE 488.2, query responses may be buffered for these reasons:

- When the query is parsed by the oscilloscope.
- When the computer addresses the oscilloscope to talk so that it may read the response.

This oscilloscope buffers responses to a query when the query is parsed.

## EOI

The EOI bus control line follows the IEEE 488.2 standard without exception.

# 6

# Sample Programs

# Sample Programs

Sample programs for the Infiniium-Series Oscilloscopes are shipped on a CD ROM with the instrument. Each program demonstrates specific sets of instructions.

This chapter shows you some of those functions, and describes the commands being executed. Both C and BASIC examples are included.

The header file is:

- gpibdecl.h

The C examples include:

- init.c
- gen_srq.c
- srqagi.c
- srqnat.c
- learnstr.c
- sicl_IO.c
- natl_IO.c

The BASIC examples include:

- init.bas
- srq.bas
- lrn_str.bas

The sample program listings are included at the end of this chapter.

# Sample Program Structure

This chapter includes segments of both the C and BASIC sample programs. Each program includes the basic functions of initializing the interface and oscilloscope, capturing the data, and analyzing the data.

In general, both the C and BASIC sample programs typically contain the following fundamental segments:

| Segment | Description |
|---|---|
| main program | Defines global variables and constants, specifies include files, and calls various functions. |
| initialize | Initializes the GPIB or LAN interface and oscilloscope, and sets up the oscilloscope and the ACQuire subsystem. |
| acquire_data | Digitizes the waveform to capture data. |
| auto_measurements | Performs simple parametric measurements. |
| transfer_data | Brings waveform data and voltage/timing information (the preamble) into the computer. |

# Sample C Programs

Segments of the sample programs "init.c" and "gen_srq.c" are shown and described in this chapter.

**init.c - Initialization**

```
/* init. c */

/* Command Order Example.  This program demonstrates the order of commands
   suggested for operation of the 548xx oscilloscope via GPIB.
   This program initializes the oscilloscope, acquires data, performs
   automatic measurements, and transfers and stores the data on the
   PC as time/voltage pairs in a comma-separated file format useful
   for spreadsheet applications. It assumes a SICL INTERFACE exists
   as 'hpib7' and an 548xx oscilloscope at address 7.
   It also requires the cal waveform attached to Channel 1.

   See the README file on the demo disk for development and linking information.
*/

#include <stdio.h>          /* location of: printf() */
#include <stdlib.h>         /* location of: atof(), atoi() */
#include "gpibdecl.h"       /* prototypes, global declarations, constants */

void initialize( );         /* initialize the oscilloscope */
void acquire_data( );       /* digitize waveform */
void auto_measurements( ); /* perform built-in automatic measurements */
void transfer_data( );     /* transfers waveform data from oscilloscope to PC */
void convert_data( );       /* converts data to time/voltage values */
void store_csv( );          /* stores time/voltage pairs to comma-separated
                            /* variable file format */
```

The include statements start the program. The file "gpibdecl.h" includes prototypes and declarations that are necessary for the Infiniium Oscilloscope sample programs.

This segment of the sample program defines the functions, in order, that are used to initialize the oscilloscope, digitize the data, perform measurements, transfer data from the oscilloscope to the PC, convert the digitized data to time and voltage pairs, and store the converted data in comma-separated variable file format.

See the following descriptions of the program segments.

### init.c - Global Definitions and Main Program

```
/* GLOBALS */
int count;
double xorg,xref,xinc;          /* values necessary for conversion of data */
double yorg,yref,yinc;
int Acquired_length;
char data[MAX_LENGTH];          /* data buffer */
double time_value[MAX_LENGTH]; /* time value of data */
double volts[MAX_LENGTH];       /* voltage value of data */


void main( void )
{
/* initialize interface and device sessions */
/* note: routine found in sicl_IO.c or natl_IO.c  */

   init_IO( );

   /* initialize the oscilloscope and interface and set up SRQ */
   initialize( );
   acquire_data( );      /* capture the data */

   /* perform automated measurements on acquired data */
   auto_measurements( );
   transfer_data( );   /* transfer waveform data to the PC from oscilloscope */
   convert_data( );      /* convert data to time/voltage pairs */
   store_csv( );         /* store the time/voltage pairs as csv file */
   close_IO( );          /* close interface and device sessions */
                         /* note: routine found in sicl_IO.c or natl_IO.c  */
} /* end main() */
```

The init_IO routine initializes the oscilloscope and interface so that the oscilloscope can capture data and perform measurements on the data. At the start of the program, global symbols are defined which will be used to store and convert the digitized data to time and voltage values.

### init.c - Initializing the Oscilloscope

```
/*
*  Function name:  initialize
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the oscilloscope for proper
*  acquisition of data. The instrument is reset to a known state and the
*  interface is cleared. System headers are turned off to allow faster
*  throughput and immediate access to the data values requested by queries.
*  The oscilloscope time base, channel, and trigger subsystems are then
*  configured. Finally, the acquisition subsystem is initialized.
*/
void initialize(  )
{
   write_IO("*RST");       /* reset oscilloscope - initialize to known state */
   write_IO("*CLS");        /* clear status registers and output queue */

   write_IO(":SYSTem:HEADer OFF"); /* turn off system headers */

   /* initialize time base parameters to center reference, */
   /* 2 ms full-scale (200 us/div), and 20 us delay */
   write_IO(":TIMebase:REFerence CENTer;RANGe 2e-3;POSition 20e-6");


   /* initialize Channel1 1.6V full-scale (200 mv/div); offset -400mv */
   write_IO(":CHANnel1:RANGe 1.6;OFFSet -400e-3");


   /* initialize trigger info: channel1 waveform on positive slope at 300mv */
   write_IO(":TRIGger:EDGE:SOURce CHANnel1;SLOPe POSitive");
   write_IO(":TRIGger:LEVel CHANnel1,-0.40");


   /* initialize acquisition subsystem */
   /* Real time acquisition - no averaging; record length 4096 */
   write_IO(":ACQuire:MODE RTIMe;AVERage OFF;POINts 4096");

}  /* end initialize() */
```

### init.c - Acquiring Data

```
/*
*  Function name:  acquire_data
*  Parameters: none
*  Return value:  none
*  Description:  This routine acquires data according to the current
*  instrument settings.
*/
void acquire_data( )
{
/*
*  The root level :DIGitize command is recommended for acquisition of new
*  data. It will initialize data buffers, acquire new data, and ensure that
*  acquisition criteria are met before acquisition of data is stopped.  The
*  captured data is then available for measurements, storage, or transfer
*  to a PC.  Note that the display is automatically turned off by the
*  :DIGitize command and must be turned on to view the captured data.
*/

   write_IO(":DIGitize CHANnel1");
   write_IO(":CHANnel1:DISPlay ON"); /* turn on channel 1 display which is */
                                     /* turned off by the :DIGitize command */

} /*  end acquire_data() */
```

### init.c - Making Automatic Measurements

```
/*
*  Function name:  auto_measurements
*  Parameters:  none
*  Return value:  none
*  Description:  This routine performs automatic measurements of volts
*  peak-to-peak and frequency on the acquired data. It also demonstrates
*  two methods of error detection when using automatic measurements.
*/

void auto_measurements( )
{
float frequency, vpp;
unsigned char vpp_str[16];
unsigned char freq_str[16];
int bytes_read;

/*
*  Error checking on automatic measurements can be done using one of two methods.
*  The first method requires that you turn on results in the Measurements
*  subsystem using the command :MEASure:SEND ON.  When this is on, the oscilloscope
*  will return the measurement and a result indicator.  The result flag is zero
*  if the measurement was successfully completed, otherwise a non-zero value is
*  returned which indicates why the measurement failed. See the Programmer's
Manual
*  for descriptions of result indicators.
*
*  The second method simply requires that you check the return value of the
*  measurement.  Any measurement not made successfully will return with the value
*  +9.999E37. This could indicate that either the measurement was unable to be
*  performed, or that insufficient waveform data was available to make the
*  measurement.
*/
/*
*  METHOD ONE - turn on results to indicate whether the measurement completed
*  successfully.  Note that this requires transmission of extra data from the
*  oscilloscope.
*/
write_IO(":MEASure:SENDvalid ON");     /* turn results on */
write_IO(":MEASure:VPP? CHANnel1");   /* query -- volts peak-to-peak channel 1*/

bytes_read = read_IO(vpp_str,16L);     /* read in value and result flag */

if (vpp_str[bytes_read-2] != '0')
   printf("Automated vpp measurement error with result %c\n",
      vpp_str[bytes_read-2]);
else
   printf("VPP is %f\n",(float)atof(vpp_str));
```

```
write_IO(":MEASure:FREQuency? CHANnel1");   /* frequency channel 1 */

bytes_read = read_IO(freq_str,16L);         /* read in value and result flag */

if (freq_str[bytes_read-2] != '0')
   printf("Automated frequency measurement error with result %c\n",
   freq_str[bytes_read-2]);
else
   printf("Frequency is %f\n",(float)atof(freq_str));

/*
*  METHOD TWO - perform automated measurements and error checking with
*  :MEAS:RESULTS OFF
*/
frequency =(float)0;
vpp = (float)0;

/* turn off results */
write_IO(":MEASure:SENDvalid OFF");

write_IO(":MEASure:FREQuency? CHANnel1");  /* frequency channel 1 */
bytes_read = read_IO(freq_str,16L);        /* read in value and result flag */

frequency = (float) atof(freq_str);

if ( frequency > 9.99e37 )
   printf("\nFrequency could not be measured.\n");
else
   printf("\nThe frequency of channel 1 is %f Hz.\n", frequency );

write_IO(":MEASure:VPP? CHANnel1");
bytes_read = read_IO( vpp_str,16L );

vpp = (float) atof(vpp_str);

if ( vpp > 9.99e37 )
   printf("Peak-to-peak voltage could not be measured.\n");
else
   printf("The voltage peak-to-peak is %f volts.\n", vpp );

} /* end auto_measurements() */
```

### init.c - Error Checking

```
/* Error checking on automatic measurements can be done using one of two methods.
*  The first method requires that you turn on results in the Measurements
* subsystem using the command :MEASure:SEND ON.  When this is on, the oscilloscope
*  will return the measurement and a result indicator.  The result flag is zero
*  if the measurement was successfully completed, otherwise a non-zero value is
*  returned which indicates why the measurement failed. See the Programmer's
Manual
*  for descriptions of result indicators.

*  The second method simply requires that you check the return value of the
*  measurement. Any measurement not made successfully will return with the value
*  +9.999E37. This could indicate that either the measurement was unable to be
*  performed, or that insufficient waveform data was available to make the
*  measurement.


*  METHOD ONE - turn on results to indicate whether the measurement completed
*  successfully. Note that this requires transmission of extra data from the
*  oscilloscope. */
   write_IO(":MEASure:SENDvalid ON");       /* turn results on */

   /* query -- volts peak-to-peak channel 1*/
   write_IO(":MEASure:VPP? CHANnel1");

  bytes_read = read_IO(vpp_str,16L);       /* read in value and result flag */

   if (vpp_str[bytes_read-2] != '0')
      printf("Automated vpp measurement error with result %c\n",
      vpp_str[bytes_read-2]);
   else
      printf("VPP is %f\n",(float)atof(vpp_str));

   write_IO(":MEASure:FREQuency? CHANnel1");  /* frequency channel 1 */
  bytes_read = read_IO(freq_str,16L);       /* read in value and result flag */

   if (freq_str[bytes_read-2] != '0')
      printf("Automated frequency measurement error with result %c\n",
      freq_str[bytes_read-2]);
   else
      printf("Frequency is %f\n",(float)atof(freq_str));
```

```
/*
*  METHOD TWO - perform automated measurements and error checking with
*  :MEAS:RESULTS OFF.
*/
frequency =(float)0;
vpp = (float)0;

  /* turn off results */
   write_IO(":MEASure:SENDvalid OFF");

   write_IO(":MEASure:FREQuency? CHANnel1"); /* frequency channel 1 */
   bytes_read = read_IO(freq_str,16L);       /* read in value and result flag */

   frequency = (float) atof(freq_str);

   if ( frequency > 9.99e37 )
      printf("\nFrequency could not be measured.\n");
   else
      printf("\nThe frequency of channel 1 is %f Hz.\n", frequency );

   write_IO(":MEASure:VPP? CHANnel1");
   bytes_read = read_IO( vpp_str,16L );

   vpp = (float) atof(vpp_str);

   if ( vpp > 9.99e37 )
      printf("Peak-to-peak voltage could not be measured.\n");
   else
      printf("The voltage peak-to-peak is %f volts.\n", vpp );

} /* end auto_measurements() */
```

### init.c - Transferring Data to the PC

```
/*
*  Function name:  transfer_data
*  Parameters:  none
*  Return value:  none
*  Description:  This routine transfers the waveform conversion factors and
*  waveform data to the PC.
*/

void transfer_data(  )
{
   int header_length;
   char header_str[8];
   char term;

   char xinc_str[32],xorg_str[32],xref_str[32];
   char yinc_str[32],yref_str[32],yorg_str[32];

   int bytes_read;

   /* waveform data source channel 1 */
   write_IO (":WAVeform:SOURce CHANnel1");

   /* setup transfer format */
   write_IO(":WAVeform:FORMat BYTE");

   /* request values to allow interpretation of raw data */
   write_IO(":WAVeform:XINCrement?");

   bytes_read = read_IO(xinc_str,32L);
   xinc = atof(xinc_str);

   write_IO(":WAVeform:XORigin?");
   bytes_read = read_IO(xorg_str,32L);
   xorg = atof(xorg_str);

   write_IO(":WAVeform:XREFerence?");
   bytes_read = read_IO(xref_str,32L);
   xref = atof(xref_str);

   write_IO(":WAVeform:YINCrement?");
   bytes_read = read_IO(yinc_str,32L);
   yinc = atof(yinc_str);



   write_IO(":WAVeform:YORigin?");
   bytes_read = read_IO(yorg_str,32L);
```

```
    yorg = atof(yorg_str);

    write_IO(":WAVeform:YREFerence?");
    bytes_read = read_IO(yref_str,32L);
    yref = atof(yref_str);

    write_IO(":WAVeform:DATA?");                    /* request waveform data */
    while (data[0] != '#')
       bytes_read = read_IO(data,1L);               /* find the # character */
    bytes_read = read_IO(header_str,1L);          /* input byte counter */
    header_length = atoi(header_str);

    /* read number of points - value in bytes */
    bytes_read = read_IO(header_str,(long)header_length);

    Acquired_length = atoi(header_str);             /* number of bytes */

    bytes_read = read_IO(data,Acquired_length);     /* input waveform data */
    bytes_read = read_IO(&term,1L);           /* input termination character */

} /* end transfer_data() */
```

An example header resembles the following when the information is stripped off:

```
#510225
```

The left most "5" defines the number of digits that follow (10225). The number "10225" is the number of points in the waveform. The information is stripped off of the header to get the number of data bytes that need to be read from the oscilloscope.

### init.c - Converting Waveform Data

```
/*
*  Function name: convert_data
*  Parameters:    none
*  Return value:  none
*  Description:  This routine converts the waveform data to time/voltage
*  information using the values that describe the waveform.  These values are
*  stored in global arrays for use by other routines.
*/

void convert_data( )
{
   int i;

   for (i = 0; i < Acquired_length; i++)
   {
      time_value[i] =((i - xref) * xinc) + xorg;      /* calculate time info */
      volts[i] = ((data[i] - yref) * yinc) + yorg;    /* calculate volt info */
   }
} /* end convert_data() */
```

The data values are returned as digitized samples (sometimes called quantization levels or q-levels). These data values must be converted into voltage and time values.

### init.c - Storing Waveform Time and Voltage Information

```
/*
*  Function name:  store_csv
*  Parameters:  none
*  Return value:  none
*  Description:  This routine stores the time and voltage information about
*  the waveform as time/voltage pairs in a comma-separated variable file
*  format.
*/

void store_csv()
{
   FILE *fp;
   int i;

   fp = fopen("pairs.csv","wb");   /* open file in binary mode - clear file */
                                   /* if already exists */
   if (fp != NULL)
   {
      for (i = 0; i < Acquired_length; i++)
   {
         /* write time,volt pairs to file */
         fprintf( fp,"%e,%lf\n",time_value[i],volts[i]);
   }
      fclose( fp );                /* close file */
   }
   else
      printf("Unable to open file 'pairs.csv'\n");

}  /* end store_csv() */
```

The time and voltage information of the waveform is stored in integer format, with the time stored first, followed by a comma, and the voltage stored second.

### Sample C Program - Generating a Service Request

Segments of the sample C program "gen_srq.c" show how to initialize the interface and oscilloscope, and generate a service request.

Two include statements start the "gen_srq.c" program. The file "stdio.h" defines the standard location of the printf routine, and is needed whenever input or output functions are used. The file "gpibdecl.h" includes necessary prototypes and declarations for the Infiniium-Series Oscilloscopes sample programs. The path of these files must specify the disk drive and directory where the "include" files reside.

```
/* gen_srq.c */

/*
*  This example program initializes the 548xx oscilloscope, runs an autoscale,
*  then generates and responds to a Service Request from the oscilloscope. The
*  program assumes an 548xx at address 7, an interface card at interface select
*  code 7, and a waveform source attached to channel 1.
*/

#include <stdio.h>          /* location of: printf() */
#include "gpibdecl.h"

void initialize();
void setup_SRQ();
void create_SRQ();

void main( void )
{
   init_IO( );          /* initialize interface and device sessions */
   initialize( );       /* initialize the oscilloscope and interface */
   setup_SRQ( );       /* enable SRQs on oscilloscope and set up SRQ handler */
   create_SRQ( );       /* generate SRQ */
   close_IO( );         /* close interface and device sessions */

} /* end main() */
```

The routine "init_IO" contains three subroutines that initialize the oscilloscope and interface, and sets up and generate a service request.

The following segment describes the initialize subroutine.

### Initializing the Oscilloscope

The following function is demonstrated in the "gen_srq.c" sample program.

```
/*
*  Function name: initialize
*  Parameters:    none
*  Return value:  none
* Description: This routine initializes the oscilloscope for proper acquisition
*  of data. The instrument is reset to a known state and the interface is
*  cleared. System headers are turned off to allow faster throughput and
*  immediate access to the data values requested by queries. The oscilloscope
*  performs an autoscale to acquire waveform data.
*/

void initialize(  )
{
   write_IO("*RST");   /* reset oscilloscope - initialize to known state */
   write_IO("*CLS");   /* clear status registers and output queue */
   write_IO(":SYSTem:HEADer OFF");  /* turn off system headers */
   write_IO(":AUToscale");          /* perform autoscale */

} /* end initialize() */
```

The *RST command is a common command that resets the oscilloscope to a known default configuration. Using *RST ensures that the oscilloscope is in a known state before you configure it. It ensures very consistent and repeatable results. Without *RST, a program may run one time, but it may give different results in following runs if the oscilloscope is configured differently.

For example, if the trigger mode is normally set to edge, the program may function properly. But, if someone puts the oscilloscope in the advanced TV trigger mode from the front panel, the program may read measurement results that are totally incorrect. So, *RST defaults the oscilloscope to a set configuration so that the program can proceed from the same state each time.

The *CLS command clears the status registers and the output queue.

AUToscale finds and displays all waveforms that are attached to the oscilloscope. You should program the oscilloscope's time base, channel, and trigger for the specific measurement to be made, as you would do from the front panel, and use whatever other commands are needed to configure the oscilloscope for the desired measurement.

### Setting Up a Service Request

The following code segment shows how to generate a service request. The following function is demonstrated in the "gen_srq.c" sample program.

```
/*
*  Function name: setup_SRQ
*  Parameters:    none
*  Return value:  none
* Description: This routine initializes the device to generate Service Requests.
It
*  sets the Service Request Enable Register Event Status Bit and the Standard
*  Event Status Enable Register to allow SRQs on Command, Execution, Device
*  Dependent, or Query errors.
*/
void setup_SRQ(  )
{
   /* Enable Service Request Enable Register - Event Status Bit */

   write_IO("*SRE 32");    /* Enable Standard Event Status Enable Register */
                           /* enable Command Error - bit 4 - value 32 */
   write_IO("*ESE 32");

} /* end setup_SRQ( ) */
```

### Generating a Service Request

The following function is demonstrated in the "gen_srq.c" sample program.

```
/*
*  Function name: create_SRQ
*  Parameters:    none
*  Return value:  none
* Description: This routine sends two illegal commands to the oscilloscope which
*  will generate an SRQ and will place two error strings in the error queue.  The
*  oscilloscope ID is requested to allow time for the SRQ to be generated.  The ID
*   string will contain a leading character which is the response placed in
*   the output queue by the interrupted query.
*/

void create_SRQ( )
{
   char buf[256] = { 0 };  // read buffer for id string
   int bytes_read = 0;

#ifdef AGILENT
   // Setup the Agilent interrupt handler
   ionsrq( scope, srq_agilent );
#else
   // Setup the National interrup handler
   ibnotify( scope, RQS, srq_national, NULL );
#endif

   // Generate command error - send illegal header
   write_IO(":CHANnel:DISPlay OFF");

   srq_asserted = TRUE;

   while( srq_asserted )
   {
      // Do nothing until the interrupt has finished
   }

} /* end create_SRQ() */
```

# Listings of the Sample Programs

Listings of the C sample programs in this section include:

- gpibdecl.h
- srqagi.c
- learnstr.c
- sicl_IO.c
- natl_IO.c

Listings of the BASIC sample programs in this section include

- init.bas
- srq.bas
- lrn_str.bas

**Read the README File Before Using the Sample Programs**

Before using the sample programs, be sure to read the README file on the disk t
contains the sample programs.

## gpibdecl.h Sample Header

```
/* gpibdecl.h */

/* This file includes necessary prototypes and declarations for the
   example programs for the Agilent 548xx */

/* User must indicate which GPIB card (Agilent or National) is being used or
   if the LAN interface is being used.
   Also, if using a National card, indicate which version of windows
   (WIN31 or WIN95) is being used */


#define LAN    /* Uncomment if using LAN interface */
#define AGILENT/* Uncomment if using LAN or Agilent interface card */
// #define NATL   /* Uncomment if using National interface card */

/* #define WIN31 */    /* For National card ONLY - select windows version */
#define WIN95


#ifdef WIN95
#include <windows.h>/* include file for Windows 95 */
#else
#include <windecl.h>/* include file for Windows 3.1 */
#endif

#ifdef AGILENT
   #include "d:\siclnt\c\sicl.h" /* Change the path for the sicl.h location */
#else
   #include "decl-32.h"
#endif

#define CME 32
#define EXE 16
#define DDE  8
#define QYE  4

#define SRQ_BIT 64
#define MAX_LRNSTR 40000
#define MAX_LENGTH 4096
#define MAX_INT 4192

#ifdef AGILENT
   #ifdef LAN
      #define INTERFACE "lan[130.29.71.82]:inst0"
   #else
```

```
      #define DEVICE_ADDR "hpib7,7"
      #define INTERFACE "hpib7"
   #endif
#else
   #define INTERFACE "gpib0"

   #define board_index 0
   #define prim_addr 7
   #define second_addr 0
   #define timeout 13
   #define eoi_mode  1
   #define eos_mode 0
#endif

/*  GLOBALS */
#ifdef AGILENT
   INST bus;
   INST scope;
#else
   int bus;
   int scope;
#endif

#define TRUE 1
#define FALSE 0

extern int srq_asserted;

/* GPIB prototypes */
void init_IO( );
void write_IO( char* );
void write_lrnstr( char*, long );
int read_IO( char*, unsigned long );
unsigned char read_status( );
void close_IO();
void gpiberr();

#ifdef AGILENT
   extern void SICLCALLBACK srq_agilent( INST );
#else
   extern int __stdcall srq_national( int, int, int, long, void* );
#endif
```

# srqagi.c Sample Program

```
/* file:  srq.c */

/*  This file contains the code to handle Service Requests from an HP-IB device */


#include <stdio.h>    /* location of printf(), fopen(), and fclose() */
#include "gpibdecl.h"

int srq_asserted;

/*
* Function name: srq_agilent
*   Parameters: INST which is id of the open interface.
*   Return value: none
*   Description: This routine services the scope when an SRQ is generated.
*     An error file is opened to receive error data from the scope.
*/

void SICLCALLBACK srq_agilent( INST id )
{
   FILE *fp;
   unsigned char statusbyte = 0;
   int i =0;
   int more_errors = 0;
   char error_str[64] ={0};
   int bytes_read;

   srq_asserted = TRUE;

   statusbyte = read_status( );

   if ( statusbyte & SRQ_BIT )
   {

      fp = fopen( "error_list","wb" );                /* open error file */

      if (fp == NULL)
      printf("Error file could not be opened.\n");

      /* read error queue until no more errors */

      more_errors = TRUE;
```

```
      while ( more_errors )
      {
         write_IO(":SYSTEM:ERROR? STRING");
         bytes_read = read_IO(error_str, 64L);

         error_str[bytes_read] = '\0';
        printf("Error string:%s\n", error_str ); /* write error msg to std IO */

         if (fp != NULL)
        fprintf(fp,"Error string:%s\n", error_str ); /* write error msg to file */

            if ( error_str[0] == '0' )
           {
                write_IO("*CLS");  /* Clear event registers and queues,
                                      except output */
                more_errors = FALSE;
              if( fp != NULL)
                 fclose( fp );
           }
      } /* end while (more_errors) */
    }
    else
   {
     printf(" SRQ not generated by scope.\n ");     /* scope did not cause SRQ */
   }

      srq_asserted = FALSE;

}/* end srq_agilent */
```

# learnstr.c Sample Program

```
/* learnstr.c */

/*
*  This example program initializes the 548xx oscilloscope, runs autoscale to
*  acquire a waveform, queries for the learnstring, and stores the learnstring
*  to disk.  It then allows the user to change the setup, then restores the
*  original learnstring. It assumes that a waveform is attached to the
*  oscilloscope.
*/

#include <stdio.h>        /* location of: printf(), fopen(), fclose(),
                             fwrite(),getchar */
#include "gpibdecl.h"

void initialize( );
void store_learnstring( );
void change_setup( );
void get_learnstring( );

void main( void )
{
   init_IO( );    /* initialize device and interface  */
                  /* Note: routine found in sicl_IO.c or natl_IO.c */
   /* initialize the oscilloscope and interface, and set up SRQ */
   initialize( );
   store_learnstring( ); /* request learnstring and store */
   change_setup( );      /* request user to change setup */
   get_learnstring( );   /* restore learnstring */
   close_IO( );          /* close device and interface sessions */
                         /* Note: routine found in sicl_IO.c or natl_IO.c */
} /* end main */
```

```
/*
*  Function name:  initialize
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the oscilloscope for proper
*  acquisition of data. The instrument is reset to a known state and the
*  interface is cleared. System headers are turned off to allow faster
*  throughput and immediate access to the data values requested by queries.
*  Autoscale is performed to acquire a waveform. The waveform is then
*  digitized, and the channel display is turned on following the acquisition.
*/

void initialize(  )
{
   write_IO("*RST");  /* reset oscilloscope - initialize to known state */
   write_IO("*CLS");  /* clear status registers and output queue */

   write_IO(":SYSTem:HEADer ON");/* turn on system headers */

   /* initialize Timebase parameters to center reference, 2 ms
      full-scale (200 us/div), and 20 us delay */
   write_IO(":TIMebase:REFerence CENTer;RANGe 5e-3;POSition 20e-6");

   /* initialize Channel1 1.6v full-scale (200 mv/div);
      offset -400mv */
   write_IO(":CHANnel1:RANGe 1.6;OFFSet -400e-3");

   /* initialize trigger info: channel1 waveform on positive slope
      at 300mv */
   write_IO(":TRIGger:EDGE:SOURce CHANnel1;SLOPe POSitive");
   write_IO(":TRIGger:LEVel CHANnel1,-0.40");

   /* initialize acquisition subsystem */
   /* Real time acquisition - no averaging; record length 4096 */
   write_IO(":ACQuire:MODE RTIMe;AVERage OFF;POINts 4096");

} /* end initialize() */
```

```
/*
*  Function name:  store_learnstring
*  Parameters:  none
*  Return value:  none
*  Description:  This routine requests the system setup known as a
* learnstring. The learnstring is read from the oscilloscope and stored in a file
*  called Learn2.
*/

void store_learnstring(  )
{
   FILE *fp;
   unsigned char setup[MAX_LRNSTR]={0};
   int actualcnt = 0;

   write_IO(":SYSTem:SETup?");              /* request learnstring */
   actualcnt = read_IO(setup, MAX_LRNSTR);

   fp = fopen( "learn2","wb");


   if ( fp != NULL )
   {
      fwrite( setup,sizeof(unsigned char),(int)actualcnt,fp);
      printf("Learn string stored in file Learn2\n");

      fclose( fp );
   }
   else
      printf("Error in file open\n");

}/* end store_learnstring */

/*
*  Function name:  change_setup
*  Parameters:  none
*  Return value:  none
* Description:  This routine places the oscilloscope into local mode to allow the
*  customer to change the system setup.
*/

void change_setup( )
{
   printf("Please adjust setup and press ENTER to continue.\n");
   getchar();

} /* end change_setup */
```

```
/*
*  Function name:  get_learnstring
*  Parameters:  none
*  Return value:  none
*  Description:  This routine retrieves the system setup known as a
*  learnstring from a disk file called Learn2. It then restores
*  the system setup to the oscilloscope.
*/

void get_learnstring()
{
   FILE *fp;
   unsigned char setup[MAX_LRNSTR];
   unsigned long count = 0;

   fp = fopen( "learn2","rb");

   if ( fp != NULL )
   {
      count = fread( setup,sizeof(unsigned char),MAX_LRNSTR,fp);

      fclose( fp );
   }
   write_lrnstr(setup,count);           /* send learnstring */
   write_IO(":RUN");

}/* end get_learnstring */
```

# sicl_IO.c Sample Program

```
/* sicl_IO.c */

#include <stdio.h>                /* location of: printf() */
#include <string.h>               /* location of: strlen() */
#include "gpibdecl.h"

/*  This file contains IO and initialization routines for the SICL libraries. */
/*
*  Function name:  init_IO
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the SICL environment. It sets up
*  error handling, opens both an interface and device session, sets timeout
*  values, clears the interface by pulsing IFC, and clears the instrument
*  by performing a Selected Device Clear.
*/

void init_IO( )
{
   ionerror(I_ERROR_EXIT);        /* set-up interface error handling */

   /* open interface session for verifying SRQ line */
   bus = iopen( INTERFACE );
   if ( bus == 0 )
      printf("Bus session invalid\n");

   itimeout( bus, 20000 );        /* set bus timeout to 20 sec */
   iclear( bus );                 /* clear the interface - pulse IFC */

#ifdef LAN
   scope = bus;
#else
   scope = iopen( DEVICE_ADDR );            /* open the scope device session */
   if ( scope == 0)
      printf( "Scope session invalid\n" );
   itimeout( scope, 20000 );      /* set device timeout to 20 sec */
   iclear( scope );       /* perform Selected Device Clear on oscilloscope */
#endif
} /* end init_IO */
```

```
/*
*  Function name:  write_IO
*  Parameters:  char *buffer which is a pointer to the character string to be
*  output; unsigned long length which is the length of the string to be output
*  Return value:  none
*  Description:  This routine outputs strings to the oscilloscope device session
*  using the unformatted I/O SICL commands.
*/

void write_IO( void *buffer )
{
   unsigned long actualcnt;
   unsigned long length;
   int send_end = 1;
   length = strlen( buffer );
   iwrite( scope, buffer, length, send_end, &actualcnt );

} /* end write_IO */

/*
*  Function name:  write_lrnstr
*  Parameters:  char *buffer which is a pointer to the character string to be
*  output; long length which is the length of the string to be output
*  Return value:  none
*  Description:  This routine outputs a learnstring to the oscilloscope device
*  session using the unformatted I/O SICL commands.
*/

void write_lrnstr( void *buffer, long length )
{
   unsigned long actualcnt;
   int send_end = 1;

   iwrite( scope, buffer,(unsigned long) length,
      send_end, &actualcnt );

} /* end write_lrnstr() */
```

```
/*
*  Function name:  read_IO
*  Parameters:  char *buffer which is a pointer to the character string to be
*  input; unsigned long length which indicates the max length of the string to
*  be input
*  Return value:  integer which indicates the actual number of bytes read
*  Description:  This routine inputs strings from the oscilloscope device session
*  using SICL commands.
*/

int read_IO(void *buffer,unsigned long length)
{
   int reason;
   unsigned long actualcnt;

   iread(scope,buffer,length,&reason,&actualcnt);

   return( (int) actualcnt );
}

/*
*  Function name: check_SRQ
*  Parameters:  none
*  Return value:  integer indicating if bus SRQ line was asserted
*  Description:  This routine checks for the status of SRQ on the bus and
*  returns a value to indicate the status.
*/

int check_SRQ( )
{
   int srq_asserted;

      /* check for SRQ line status */
      igpibbusstatus(bus, I_GPIB_BUS_SRQ, &srq_asserted);

   return( srq_asserted );

} /* end check_SRQ() */
```

```
/*
*  Function name:  read_status
*  Parameters:  none
*  Return value:  unsigned char indicating the value of status byte
*  Description:  This routine reads the oscilloscope status byte and returns
*  the status.
*/

unsigned char read_status( )
{
   unsigned char statusbyte;

   /* Always read the status byte from instrument */
   /* NOTE: ireadstb uses serial poll to read status byte - this
            should clear bit 6 to allow another SRQ. */

      ireadstb( scope, &statusbyte );
   return( statusbyte );

} /* end read_status() */

/*
*  Function name:  close_IO
*  Parameters:  none
*  Return value:  none
*  Description:  This routine closes device and interface sessions for the
*  SICL environment and calls the routine _siclcleanup which de-allocates
*  resources used by the SICL environment.
*/

void close_IO( )
{
   iclose( scope );    /* close device session */
   iclose( bus );      /* close interface session */

   _siclcleanup();     /* required for 16-bit applications */

} /* end close_SICL() */
```

## natl_IO.c Sample Program

```
/* natl_IO.c */

#include <stdio.h>      /* location of: printf() */
#include <string.h>     /* location of: strlen() */
#include "gpibdecl.h"

/* This file contains IO and initialization routines for the NI488.2 commands. */
/*
*  Function name:  gpiberr
*  Parameters:  char* - string describing error
*  Return value:  none
*  Description:  This routine outputs error descriptions to an error file.
*/

void gpiberr( char *buffer )
{
   printf("Error string: %s\n",buffer );

} /* end gpiberr() */

/*
*  Function name:  init_IO
*  Parameters:  none
*  Return value:  none
*  Description:  This routine initializes the NI environment. It sets up error
*  handling, opens both an interface and device session, sets timeout values
*  clears the interface by pulsing IFC, and clears the instrument by performing
*  a Selected Device Clear.
*/

void init_IO( )
{
   bus = ibfind( INTERFACE );          /* open and initialize GPIB board */
   if( ibsta & ERR )
      gpiberr("ibfind error");

   ibconfig( bus, IbcAUTOPOLL, 0);  /* turn off autopolling */

   ibsic( bus );                    /* clear interface - pulse IFC */
   if( ibsta & ERR )
   {
      gpiberr( "ibsic error" );
   }
```

```
    /* open device session */
    scope = ibdev( board_index, prim_addr, second_addr, timeout,
                   eoi_mode, eos_mode );
    if( ibsta & ERR )
    {
       gpiberr( "ibdev error" );
    }

    ibclr( scope );                    /* clear the device( scope ) */

    if( ibsta & ERR)
    {
       gpiberr("ibclr error" );
    }

} /* end init_IO */

/*
*  Function name:  write_IO
*  Parameters:  void *buffer which is a pointer to the character string
*               to be output
*  Return value:  none
* Description: This routine outputs strings to the oscilloscope device session.
*/
void write_IO( void *buffer )
{
   long length;

   length = strlen( buffer );

   ibwrt( scope, buffer, (long) length );
   if ( ibsta & ERR )
   {
      gpiberr( "ibwrt error" );
   }

} /* end write_IO() */
```

```
/*
*  Function name:  write_lrnstr
*  Parameters:  void *buffer which is a pointer to the character string to
*  be output; length which is the length of the string to be output
*  Return value:  none
*  Description:  This routine outputs a learnstring to the oscilloscope device
*  session.
*/
void write_lrnstr( void *buffer, long length )
{

   ibwrt( scope, buffer, (long) length );
   if ( ibsta & ERR )
   {
      gpiberr( "ibwrt error" );
   }

} /* end write_lrnstr() */

/*
*  Function name:  read_IO
*  Parameters:  char *buffer which is a pointer to the character string to be
*  input; unsigned long length which indicates the max length of the string
*  to be input
*  Return value:  integer which indicates the actual number of bytes read
* Description: This routine inputs strings from the oscilloscope device session.
*/

int read_IO(void *buffer,unsigned long length)
{
   ibrd(scope, buffer,( long )length );

   return( ibcntl );

} /* end read_IO() */
```

```
/*
*  Function name:  check_SRQ
*  Parameters:  none
*  Return value:  integer indicating if bus SRQ line was asserted
*  Description:  This routine checks for the status of SRQ on the bus and
*  returns a value to indicate the status.
*/

int check_SRQ( )
{
   int srq_asserted;
   short control_lines = 0;

   iblines( bus, &control_lines);

   if( control_lines & BusSRQ )
      srq_asserted = TRUE;
   else
      srq_asserted = FALSE;

    return( srq_asserted );

} /* end check_SRQ() */

/*
*  Function name:  read_status
*  Parameters:  none
*  Return value:  unsigned char indicating the value of status byte
*  Description:  This routine reads the oscilloscope status byte and returns
*  the status.
*/
unsigned char read_status( )
{
   unsigned char statusbyte;

   /* Always read the status byte from instrument */

   ibrsp( scope,  &statusbyte );

   return( statusbyte );

} /* end read_status() */
```

```
/*
*  Function name:  close_IO
*  Parameters:  none
*  Return value:  none
*  Description:  This routine closes device session.
*/

void close_IO( )
{
   ibonl( scope,0 );  /* close device session */

} /* end close_IO() */
```

# init.bas Sample Program

```
10    !file: init
20    !
30    !
40     !    This program demonstrates the order of commands suggested for
operation of
50    !    the 548xx oscilloscope via GPIB.  This program initializes the
oscilloscope, acquires
60    !    data, performs automatic measurements, and transfers and stores the
data on the
70    !    PC as time/voltage pairs in a comma-separated file format useful for
spreadsheet
80    !   applications.  It assumes an interface card at interface select code 7, an
90    !    548xx oscilloscope at address 7, and the 548xx cal waveform connected
to Channel 1.
100   !
110   !
120   !
130   COM /Io/@Scope,@Path,Interface
140   COM /Raw_data/ INTEGER Data(4095)
150   COM /Converted_data/ REAL Time(4095),Volts(4095)
160   COM /Variables/ REAL Xinc,Xref,Xorg,Yinc,Yref,Yorg
170   COM /Variables/ INTEGER Record_length
180   !
190   !
200   CALL Initialize
210   CALL Acquire_data
220   CALL Auto_msmts
230   CALL Transfer_data
240   CALL Convert_data
250   CALL Store_csv
260   CALL Close
270   END
280   !
290
!!!
```

```
360   !
370   !     Subprogram name:  Initialize
380   !     Parameters: none
390   !     Return value: none
400   !     Description:  This routine initializes the interface and the
oscilloscope.   The instrument
410   !     is reset to a known state and the interface is cleared.  System headers
420   !     are turned off to allow faster throughput and immediate access to the
430   !     data values requested by the queries. The oscilloscope time base,
440   !     channel, and trigger subsystems are then configured.  Finally, the
450   !     acquisition subsystem is initialized.
460   !
470   !
480   SUB Initialize
490   COM /Io/@Scope,@Path,Interface
500   COM /Variables/ REAL Xinc,Xref,Xorg,Yinc,Yref,Yorg
510   COM /Variables/ INTEGER Record_length
520        Interface=7
530        ASSIGN @Scope TO 707
540        RESET Interface
550        CLEAR @Scope
560        OUTPUT @Scope;"*RST"
570        OUTPUT @Scope;"*CLS"
580        OUTPUT @Scope;":SYSTem:HEADer OFF"
590        !Initialize Timebase: center reference, 2 ms full-scale (200 us/div),
               20 us delay
600       OUTPUT @Scope;":TIMebase:REFerence CENTer;RANGe 2e-3;POSition 20e-6"
610         ! Initialize Channel1:  1.6V full-scale (200mv/div), -415mv offset
620        OUTPUT @Scope;":CHANnel1:RANGe 1.6;OFFSet -415e-3"
630          !Initialize Trigger: Edge trigger, channel1 source at -415mv
640        OUTPUT @Scope;":TRIGger:EDGE:SOURce CHANnel1;SLOPe POSitive"
650        OUTPUT @Scope;":TRIGger:LEVel CHANnel1,-0.415"
660          ! Initialize acquisition subsystem
665          ! Real time acquisition, Averaging off, memory depth 4096
670        OUTPUT @Scope;":ACQuire:MODE RTIMe;AVERage OFF;POINts 4096"
680        Record_length=4096
690     SUBEND
700     !
710     !
720
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!
730     !
740     !
750     !     Subprogram name:  Acquire_data
760     !     Parameters: none
770     !     Return value: none
780     !     Description:  This routine acquires data according to the current
```

```
instrument
790     !                 setting.  It uses the root level :DIGitize command.
This command
800     !                  is recommended for acquisition of new data because
it will initialize
810     !                 the data buffers, acquire new data, and ensure that
acquisition
820     !                   criteria are met before acquisition of data is
stopped.  The captured
830     !                   data is then available for measurements, storage,
or transfer to a
840     !                  PC.  Note that the display is automatically turned
off by the :DIGitize
850     !               command and must be turned on to view the captured data.
860     !
870     !
880     SUB Acquire_data
890     COM /Io/@Scope,@Path,Interface
900     OUTPUT @Scope;":DIGitize CHANnel1"
910     OUTPUT @Scope;":CHANnel1:DISPlay ON"
920     SUBEND
930     !
940     !
950
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!
960     !
970     !
980     !     Subprogram name:  Auto_msmts
990     !     Parameters: none
1000    !     Return value: none

1010    !     Description:  This routine performs automatic measurements of
volts peak-to-peak
1020    !                   and frequency on the acquired data.  It also
demonstrates two methods
1030    !                 of error detection when using automatic measurements.
1040    !
1050    !
1060    SUB Auto_msmts
1070    COM /Io/@Scope,@Path,Interface
1080    REAL Freq,Vpp
1090    DIM Vpp_str$[64]
1100    DIM Freq_str$[64]
1110    Bytes_read=0
1120    !
1130    !     Error checking on automatic measurements can be done using one of
two methods.
```

```
1140   !    The first method requires that you turn on results in the Measurement
subsystem
1150   !      using the command ":MEASure:SEND ON".  When this is on, the
oscilloscope will return the
1160   !      measurement and a result indicator.  The result flag is zero if
the measurement
1170   !    was successfully completed, otherwise a non-zero value is returned
which indicates
1180   !      why the measurement failed.  See the Programmer's Manual for
descriptions of result
1190   !     indicators.  The second method simply requires that you check the
return value of
1200   !    the measurement.  Any measurement not made successfully will return
with the value
1210   !      +9.999e37.  This could indicate that either the measurement was
unable to be
1220   !    performed or that insufficient waveform data was available to make
the measurement.
1230   !
1240   !      METHOD ONE
1250   !
1260         OUTPUT @Scope;":MEASure:SENDvalid ON"       !turn on results
1270       OUTPUT @Scope;":MEASure:VPP? CHANnel1"    !Query volts peak-to-peak
1280         ENTER @Scope;Vpp_str$
1290        Bytes_read=LEN(Vpp_str$)                   !Find length of string
1300         CLEAR SCREEN
1310         IF Vpp_str$[Bytes_read;1]="0" THEN         !Check result value
1320            PRINT
1330            PRINT "VPP is ";VAL(Vpp_str$[1,Bytes_read-1])
1340            PRINT
1350         ELSE
1360            PRINT
1370            PRINT "Automated vpp measurement error with result
";Vpp_str$[Bytes_read;1]
1380            PRINT
1390         END IF
1400   !
1410   !
1420         OUTPUT @Scope;":MEASure:FREQuency? CHANnel1"  !Query frequency
1430         ENTER @Scope;Freq_str$
1440        Bytes_read=LEN(Freq_str$)                    !Find string length
1450       IF Freq_str$[Bytes_read;1]="0" THEN         !Determine result value
1460            PRINT
1470            PRINT "Frequency is ";VAL(Freq_str$[1,Bytes_read-1])
1480            PRINT
1490         ELSE
1500            PRINT
1510            PRINT "Automated frequency measurement error with result
```

```
";Freq_str$[Bytes_read;1]
1520              PRINT
1530           END IF
1540     !
1550     !
1560     !      METHOD TWO
1570     !
1580           OUTPUT @Scope;":MEASure:SENDvalid OFF"      !turn off results
1590         OUTPUT @Scope;":MEASure:VPP? CHANnel1"     !Query volts peak-to-peak
1600           ENTER @Scope;Vpp
1610           IF Vpp<9.99E+37 THEN
1620              PRINT
1630              PRINT "VPP is ";Vpp
1640              PRINT
1650           ELSE
1660              PRINT
1670              PRINT "Automated vpp measurement error ";Vpp
1680              PRINT
1690           END IF
1700           OUTPUT @Scope;":MEASure:FREQuency? CHANnel1"
1710           ENTER @Scope;Freq
1720           IF Freq<9.99E+37 THEN
1730              PRINT
1740              PRINT "Frequency is ";Freq
1750              PRINT
1760           ELSE
1770              PRINT
1780              PRINT "Automated frequency measurement error";Freq
1790              PRINT
1800           END IF
1810    SUBEND
1820    !
1830    !
1840
!!!
```

```
1960    COM /Raw_data/ INTEGER Data(4095)
1970    COM /Converted_data/ REAL Time(4095),Volts(4095)
1980    COM /Variables/ REAL Xinc,Xref,Xorg,Yinc,Yref,Yorg
1990    COM /Variables/ INTEGER Record_length
2000    !                                  define waveform data source and format
2010    OUTPUT @Scope;":WAVeform:SOURce CHANnel1"
2020    OUTPUT @Scope;":WAVeform:FORMat WORD"
2030   !                          request values needed to convert raw data to real
2040    OUTPUT @Scope;":WAVeform:XINCrement?"
2050    ENTER @Scope;Xinc
2060    OUTPUT @Scope;":WAVeform:XORigin?"
2070    ENTER @Scope;Xorg
2080    OUTPUT @Scope;":WAVeform:XREFerence?"
2090    ENTER @Scope;Xref
2100    OUTPUT @Scope;":WAVeform:YINCrement?"
2110    ENTER @Scope;Yinc
2120    OUTPUT @Scope;":WAVeform:YORigin?"
2130    ENTER @Scope;Yorg
2140    OUTPUT @Scope;":WAVeform:YREFerence?"
2150    ENTER @Scope;Yref
2160    !
2170    !                                          request data
2180    OUTPUT @Scope;":WAVeform:DATA?"
2190    ENTER @Scope USING "#,1A";First_chr$       !ignore leading #
2200    ENTER @Scope USING "#,1D";Header_length    !input number of bytes in
header value
2210   ENTER @Scope USING "#,"&VAL$(Header_length)&"D";Record_length    !Record
length in bytes
2220    Record_length=Record_length/2              !Record length in words
2230    ENTER @Scope USING "#,W";Data(*)
2240    ENTER @Scope USING "#,A";Term$             !Enter terminating character
2250    !
2260    SUBEND
2270    !
2280    !
2290
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!
2300    !
2310    !
2320    !     Subprogram name:  Convert_data
2330    !     Parameters: none
2340    !     Return value: none
2350    !     Description:  This routine converts the waveform data to time/
voltage information
2360    !                   using the values Xinc, Xref, Xorg, Yinc, Yref, and
Yorg used to describe
2370    !                   the raw waveform data.
```

```
2380   !
2390   !
2400   SUB Convert_data
2410   COM /Io/@Scope,@Path,Interface
2420   COM /Raw_data/ INTEGER Data(4095)
2430   COM /Converted_data/ REAL Time(4095),Volts(4095)
2440   COM /Variables/ REAL Xinc,Xref,Xorg,Yinc,Yref,Yorg
2450   COM /Variables/ INTEGER Record_length
2460   !
2470   FOR I=0 TO Record_length-1
2480        Time(I)=(((I)-Xref)*Xinc)+Xorg
2490        Volts(I)=((Data(I)-Yref)*Yinc)+Yorg
2500   NEXT I
2510   SUBEND
2520   !
2530   !
2540
!!!
```

```
2820
!!!
```

# srq.bas Sample Program

```
10    !File: srq.bas
20    !
30    !  This program demonstrates how to set up and check Service Requests from
40    !   the oscilloscope.  It assumes an interface select code of 7 with an
oscilloscope at
50    !  address 7.  It also assumes a waveform is connected to the oscilloscope.
60    !
70    !
80    COM /Io/@Scope,Interface
90    COM /Variables/Temp
100   CALL Initialize
110   CALL Setup_srq
120     ON INTR Interface CALL Srq_handler   !Set up routine to handle interrupt
130      ENABLE INTR Interface;2             !Enable SRQ Interrupt for Interface
140   CALL Create_srq
150   CALL Close
160   END
170   !
180
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
190   !
200   !                     BEGIN SUBPROGRAMS
210   !
220
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
230   !
240   !
250   !     Subprogram name:  Initialize
260   !     Parameters: none
270   !     Return value: none
280   !     Description:   This routine initializes the interface and the
oscilloscope.
290   !            The instrument is reset to a known state and the interface is
300   !                    cleared.  System headers are turned off to allow
faster throughput
310   !           and immediate access to the data values requested by the queries.
320   !
330   !
340   SUB Initialize
350   COM /Io/@Scope,Interface
360        ASSIGN @Scope TO 707
370        Interface=7
380        RESET Interface
390        CLEAR @Scope
```

```
400         OUTPUT @Scope;"*RST"
410         OUTPUT @Scope;"*CLS"
420         OUTPUT @Scope;":SYSTem:HEADer OFF"
430         OUTPUT @Scope;":AUToscale"
440    SUBEND
450    !
460    !
470    !
480
!!!
```

```
800   !               a query interrupt error.  An illegal command header is then
810   !                   sent to demonstrate how to handle multiple errors in
the error queue.
820   !
830   !
840   !
850   SUB Create_srq
860   COM /Io/@Scope,Interface
870         DIM Buf$[256]
880         OUTPUT @Scope;":CHANnel2:DISPlay?"
890         OUTPUT @Scope;":CHANnel2:DISPlay OFF"     !send query interrupt
900         OUTPUT @Scope;":CHANnel:DISPlay OFF"      !send illegal header
910                         ! Do some stuff to allow time for SRQ to be recognized
920                                 !
930         OUTPUT @Scope;"*IDN?"   !Request IDN to verify communication
940           ENTER @Scope;Buf$       !NOTE: There is a leading zero to this
query response
950         PRINT                   !which represents the response to the
interrupted query above
960         PRINT Buf$
970         PRINT
980   SUBEND
990   !
1000  !
1010  !
1020
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!
1030  !
1040  !
1050  !     Subprogram name:  Srq_handler
1060  !     Parameters: none
1070  !     Return value: none
1080  !    Description:  This routine verifies the status of the SRQ line.  It
then checks
1090  !                   the status byte of the oscilloscope to determine if
the oscilloscope caused the
1100  !                  SRQ.  Note that using a SPOLL to read the status byte
of the oscilloscope
1110  !                   clears the SRQ and allows another to be generated.
The error queue
1120  !                  is read until all errors have been cleared.  All event
registers and
1130  !                   queues, except the output queue, are cleared before
control is returned
1140  !                   to the main program.
1150  !
1160  !
```

```
1170  !
1180  SUB Srq_handler
1190       COM /Io/@Scope,Interface
1200       DIM Error_str$[64]
1210       INTEGER Srq_asserted,More_errors
1220     Status_byte=SPOLL(@Scope)
1230     IF BIT(Status_byte,6) THEN
1240           More_errors=1
1250           WHILE More_errors
1260               OUTPUT @Scope;":SYSTem:ERROR? STRING"
1270               ENTER @Scope;Error_str$
1280               PRINT
1290               PRINT Error_str$
1300               IF Error_str$[1,1]="0" THEN
1310                    OUTPUT @Scope;"*CLS"
1320                    More_errors=0
1330               END IF
1340           END WHILE
1350       ELSE
1360           PRINT
1370           PRINT "Scope did not cause SRQ"
1380           PRINT
1390       END IF
1400       ENABLE INTR Interface;2     !re-enable SRQ
1410  SUBEND
1420  !
1430  !
1440
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!
1450  !
1460  !     Subprogram name:  Close
1470  !     Parameters: none
1480  !     Return value: none
1490  !     Description:  This routine resets the interface.
1500  !
1510  !
1520  !
1530  SUB Close
1540  COM /Io/@Scope,Interface
1550
1560  RESET Interface
1570  SUBEND
1580  !
1590  !
1600
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!
```

# lrn_str.bas Sample Program

```
10    !FILE: lrn_str.bas
20    !
30    !THIS PROGRAM WILL INITIALIZE THE OSCILLOSCOPE, AUTOSCALE, AND DIGITIZE
THE WAVEFORM
40    !INFORMATION.  IT WILL THEN QUERY THE INSTRUMENT FOR THE LEARNSTRING AND WILL
50    !SAVE THE INFORMATION TO A FILE.  THE PROGRAM WILL THEN PROMPT YOU TO CHANGE
60    !THE SETUP THEN RESTORE THE ORIGINAL LEARNSTRING CONFIGURATION. IT ASSUMES
70    !AN 548xx at ADDRESS 7, GPIB INTERFACE at 7, AND THE CAL waveform ATTACHED TO
80    !CHANNEL 1.
90    !
100   !
110   COM /Io/@Scope,@Path,Interface
120   COM /Variables/Max_length
130   CALL Initialize
140   CALL Store_lrnstr
150   CALL Change_setup
160   CALL Get_lrnstr
170   CALL Close
180   END
190   !
200   !
210
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
220   !
230   !                    BEGIN SUBROUTINES
240   !
250
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
260   !    Subprogram name:  Initialize
270   !    Parameters: none
280   !    Return value: none
290   !    Description:  This routine initializes the path descriptions and
resets the
300   !                interface and the oscilloscope.  It performs an autoscale
on the waveform,
310   !                acquires the data on channel 1, and turns on the display.
320   !           NOTE:  This routine also turns on system headers.  This allows the
330   !                    string ":SYSTEM:SETUP " to be returned with the
learnstring so the
340   !                    return string is in the proper format.
350   !
360   SUB Initialize
```

```
370         COM /Io/@Scope,@Path,Interface
380         COM /Variables/Max_length
390         Max_length=14000
400         ASSIGN @Scope TO 707
410         Interface=7
420         RESET Interface
430         CLEAR @Scope
440         OUTPUT @Scope;"*RST"
450         OUTPUT @Scope;"*CLS"
460         OUTPUT @Scope;":SYSTem:HEADer ON"
470         OUTPUT @Scope;":AUToscale"
480    SUBEND
490    !
500    !
510
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!
520    !
530    !
540    !      Subprogram name: Store_lrnstr
550    !      Parameters: none
560    !      Return value: none
570    !      Description:  This routine creates a file in which to store the
learnstring
580    !            configuration (Filename:Lrn_strg). It requests the learnstring
590    !             and inputs the configuration to the PC. Finally, it stores the
600    !                 configuration to the file.
610    !
620    SUB Store_lrnstr
630         COM /Io/@Scope,@Path,Interface
640         COM /Variables/Max_length
650         ON ERROR GOTO Cont
660         PURGE "Lrn_strg"
670 Cont:   OFF ERROR
680         CREATE BDAT "Lrn_strg",1,14000
690         DIM Setup$[14000]
700         ASSIGN @Path TO "Lrn_strg"
710         OUTPUT @Scope;":SYSTem:SETup?"
720         ENTER @Scope USING "-K";Setup$
730         OUTPUT @Path,1;Setup$
740         CLEAR SCREEN
750         PRINT "Learn string stored in file: Lrn_strg"
760    SUBEND
770    !
780    !
790
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!
```

```
800   !
810   !     Subprogram name: Change_setup
820   !     Parameters: none
830   !     Return value: none
840   !     Description:  This subprogram requests that the user change the
850   !                   oscilloscope setup, then press a key to continue.
860   !
870   !
880   SUB Change_setup
890        COM /Io/@Scope,@Path,Interface
900
910        PRINT
920        PRINT "Please adjust setup and press Continue to resume."
930        PAUSE
940   SUBEND
950   !
960   !
970
!!!
```

```
1230  !
1240  !
1250  !
1260  SUB Close
1270  COM /Io/@Scope,@Path,Interface
1280
1290  RESET Interface
1300  ASSIGN @Path TO *
1310  SUBEND
1320  !
1330
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!
```

7

# Common Commands

# Common Commands

Common commands are defined by the IEEE 488.2 standard. They control generic device functions that are common to many different types of instruments. Common commands can be received and processed by the oscilloscope, whether they are sent over the GPIB as separate program messages or within other program messages.

These common commands and queries are implemented in the Infiniium Oscilloscopes:

- *CLS (Clear Status)
- *ESE (Event Status Enable)
- *ESR? (Event Status Register)
- *IDN? (Identification Number)
- *LRN? (Learn)
- *OPC (Operation Complete)
- *OPT? (Option)
- *PSC (Power-on Status Clear)
- *RCL (Recall)
- *RST (Reset)
- *SAV (Save)
- *SRE (Service Request Enable)
- *STB? (Status Byte)
- *TRG (Trigger)
- *TST? (Test)
- *WAI (Wait-to-Continue)

### Receiving Common Commands

Common commands can be received and processed by the oscilloscope, whether they are sent over the GPIB as separate program messages or within other program messages. If a subsystem is currently selected and a common command is received by the oscilloscope, the oscilloscope remains in the selected subsystem. For example, if the program message

```
"ACQUIRE:AVERAGE ON;*CLS;COUNT 1024"
```

is received by the oscilloscope, the oscilloscope sets the acquire type, clears the status information, then sets the number of averages without leaving the selected subsystem.

**Headers and Common Commands.**

Headers are not prepended to common commands.

### Status Registers

The following two status registers used by common commands have an enable (mask) register. By setting bits in the enable register, you can select the status information for use. Refer to the chapter, "Status Reporting," for a complete discussion of status.

**Table 7-1 Status and Enable Registers**

| Status Register | Enable Register |
|---|---|
| Event Status Register | Event Status Enable Register |
| Status Byte Register | Service Request Enable Register |

# *CLS (Clear Status)

**Command**

`*CLS`

The *CLS command clears all status and error registers.

**Example**

This example clears the status data structures of the oscilloscope.

```
10  OUTPUT 707;"*CLS"
20  END
```

**See Also**

Refer to the "Status Reporting" chapter for a complete discussion of status.

# *ESE (Event Status Enable)

**Command**

```
*ESE <mask>
```

The *ESE command sets the Standard Event Status Enable Register bits.

`<mask>` An integer, 0 to 255, representing a mask value for the bits to be enabled in the Standard Event Status Register as shown in Table 7-2.

**Example**

This example enables the User Request (URQ) bit of the Standard Event Status Enable Register. When this bit is enabled and a front-panel key is pressed, the Event Summary bit (ESB) in the Status Byte Register is also set.

```
10  OUTPUT 707;"*ESE 64"
20  END
```

**Query**

```
*ESE?
```

The *ESE? query returns the current contents of the Standard Event Status Enable Register.

**Returned Format**

```
<mask><NL>
```

`<mask>` An integer, +0 to +255 (the plus sign is also returned), representing a mask value for the bits enabled in the Standard Event Status Register as shown in Table 7-2.

**Example**

This example places the current contents of the Standard Event Status Enable Register in the numeric variable, Event. The value of the variable is printed on the computer's screen.

```
10  OUTPUT 707;"*ESE?"
20  ENTER 707;Event
30  PRINT Event
40  END
```

The Standard Event Status Enable Register contains a mask value for the bits to be enabled in the Standard Event Status Register. A "1" in the Standard Event Status Enable Register enables the corresponding bit in the Standard Event Status Register. A "0" in the enable register disables the corresponding bit.

**Table 7-2** **Standard Event Status Enable Register Bits**

| Bit | Weight | Enables | Definition |
|---|---|---|---|
| 7 | 128 | PON - Power On | Indicates power is turned on. |
| 6 | 64 | | Not Used.<br>Permanently set to zero. |
| 5 | 32 | CME - Command Error | Indicates whether the parser detected an error. |
| 4 | 16 | EXE - Execution Error | Indicates whether a parameter was out of range, or was inconsistent with the current settings. |
| 3 | 8 | DDE - Device Dependent Error | Indicates whether the device was unable to complete an operation for device-dependent reasons. |
| 2 | 4 | QYE - Query Error | Indicates if the protocol for queries has been violated. |
| 1 | 2 | RQC - Request Control | Indicates whether the device is requesting control. |
| 0 | 1 | OPC - Operation Complete | Indicates whether the device has completed all pending operations. |

**See Also** Refer to the chapter, "Status Reporting," for a complete discussion of status.

# *ESR? (Event Status Register)

Query

```
*ESR?
```

The *ESR? query returns the contents of the Standard Event Status Register. Reading this register clears the Standard Event Status Register, as does a *CLS.

Returned Format

```
<status><NL>
```

`<status>` An integer, 0 to 255, representing the total bit weights of all bits that are high at the time you read the register.

**Example**

This example places the current contents of the Standard Event Status Register in the numeric variable, Event, then prints the value of the variable to the computer's screen.

```
10  OUTPUT 707;"*ESR?"
20  ENTER 707;Event
30  PRINT Event
40  END
```

Table 7-3 lists each bit in the Event Status Register and the corresponding bit weights.

**Table 7-3**

**Standard Event Status Register Bits**

| Bit | Bit Weight | Bit Name | Condition |
|---|---|---|---|
| 7 | 128 | PON | 1 = OFF to ON transition has occurred. |
| 6 | 64 | | Not Used. Permanently set to zero. |
| 5 | 32 | CME | 0 = no command errors.<br>1 = a command error has been detected. |
| 4 | 16 | EXE | 0 = no execution error.<br>1 = an execution error has been detected. |
| 3 | 8 | DDE | 0 = no device-dependent errors.<br>1 = a device-dependent error has been detected. |
| 2 | 4 | QYE | 0 = no query errors.<br>1 = a query error has been detected. |
| 1 | 2 | RQC | 0 = request control - NOT used - always 0. |
| 0 | 1 | OPC | 0 = operation is not complete.<br>1 = operation is complete. |
| | 0 = False = Low | | 1 = True = High |

# *IDN? (Identification Number)

**Query** `*IDN?`

The *IDN? query returns the company name, oscilloscope model number, serial number, and software version by returning this string:

```
HEWLETT-PACKARD,548xxA,<USXXXXXXXX>,<Rev #>
```

`<USXXXXXXXX>` Specifies the serial number of the oscilloscope. The first four digits and letter are the serial prefix, which is the same for all identical oscilloscopes. The last five digits are the serial suffix, which is assigned sequentially, and is different for each oscilloscope.

`<Rev #>` Specifies the software version of the oscilloscope, and is the revision number.

**Returned Format** `HEWLETT-PACKARD,548xxA,USXXXXXXXX,A.XX.XX<NL>`

**Example** This example places the oscilloscope's identification information in the string variable, Identify$, then prints the identification information to the computer's screen.

```
10  DIM Identify$[50]!dimension variable
20  OUTPUT 707;"*IDN?"
30  ENTER 707;Identify$
40  PRINT Identify$
50  END
```

# *LRN? (Learn)

Query

```
*LRN?
```

The *LRN? query returns a string that contains the oscilloscope's current setup. You can store the oscilloscope's setup and send it back to the oscilloscope at a later time. This setup string should be sent to the oscilloscope just as it is. It works because of its embedded ":SYSTem:SETup" header.

Returned Format

```
:SYSTem:SETup <setup><NL>
```

`<setup>` This is a definite-length, arbitrary block response specifying the current oscilloscope setup. The block size is subject to change with different firmware revisions.

**Example**

This example sets the oscilloscope's address and asks for the learn string, then determines the string length according to the IEEE 488.2 block specification. It then reads the string and the last EOF character.

```
10  ! Set up the oscilloscope's address and
20  ! ask for the learn string...
30  ASSIGN @Scope TO 707
40  OUTPUT @Scope:"*LRN?"
50  !
60  ! Search for the # sign.
70  !
80  Find_pound_sign: !
90  ENTER @Scope USING "#,A";Thischar$
100 IF Thischar$<>"#" THEN Find_pound_sign
110 !
120 ! Determine the string length according
130 ! to the IEEE 488.2 # block spec.
140 ! Read the string then the last EOF char.
150 !
160 ENTER @Scope USING "#,D";Digit_count
170 ENTER @Scope USING
"#,"&VAL$(Digit_count)&"D";Stringlength
180 ALLOCATE Learn_string$[Stringlength+1]
190 ENTER @Scope USING "-K";Learn_string$
200 OUTPUT 707;":syst:err?"
210 ENTER 707;Errornum
220 PRINT "Error Status=";Errornum
```

**See Also**

:SYSTem:SETup command and query. When HEADers and LONGform are ON, the :SYSTem:SETup command performs the same function as the *LRN? query. Otherwise, *LRN and SETup are not interchangeable.

***LRN? Returns Prefix to Setup Block**

The *LRN query always returns :SYSTem:SETup as a prefix to the setup block. The :SYSTem:HEADer command has no effect on this response.

# *OPC (Operation Complete)

Command

```
*OPC
```

The *OPC command sets the operation complete bit in the Standard Event Status Register when all pending device operations have finished.

**Example**

This example sets the operation complete bit in the Standard Event Status Register when the DIGitize operation is complete.

```
10  OUTPUT 707;":DIGITIZE CHANNEL1;*OPC"
20  END
```

Query

```
*OPC?
```

The *OPC? query places an ASCII character "1" in the oscilloscope's output queue when all pending selected device operations have finished.

Returned Format

```
1<NL>
```

**Example**

This example places an ASCII character "1" in the oscilloscope's output queue when the AUToscale operation is complete. Then the value in the output queue is placed in the numeric variable "Complete."

```
10  OUTPUT 707;":AUTOSCALE;*OPC?"
20  ENTER 707;Complete
30  PRINT Complete
40  END
```

The *OPC? query allows synchronization between the computer and the oscilloscope by using the message available (MAV) bit in the Status Byte, or by reading the output queue. Unlike the *OPC command, the *OPC query does not affect the OPC Event bit in the Standard Event Status Register.

# *OPT? (Option)

Query

```
*OPT?
```

The *OPT? query returns a string with a list of installed options. If no options are installed, the string will have a 0 as the first character.

The length of the returned string may increase as options become available in the future. Once implemented, an option name will be appended to the end of the returned string, delimited by a comma.

**Example**

This example places all options into the string variable, Options$, then prints the option model and serial numbers to the computer's screen.

```
10  DIM Options$[100]
20  OUTPUT 707;"*OPT?"
30  ENTER 707;Options$
40  PRINT Options$
50  END
```

# *PSC (Power-on Status Clear)

Command

```
*PSC {{ON|1} | {OFF|0}}
```

The *PSC command determines whether or not the SRQ line is set upon the completion of the oscilloscope's boot process. When the *PSC flag is set to 1, the Power On (PON) bit of the Standard Event Status Register is 0 during the boot process. When the *PSC flag is set to 0, the PON bit is set to a 1 during the boot process.

When the *PSC flag is set to 0, the Standard Event Status Enable Register must be set to 128 decimal and the Service Request Enable Register must be set to 32 decimal. This allows the Power On (PON) bit to set the SRQ line when the oscilloscope is ready to receive commands.

If you are using a LAN interface rather than a GPIB interface, it is not possible to receive the SRQ during the boot process.

**Example**

This example sets the *PSC flag to 0 which sets the SRQ line during the boot process.

```
10 OUTPUT 707;"*PSC 0;*SRE 32;*ESE 128"
20 END
```

Query

The *PSC? query returns the value of the *PSC flag.

Returned Format

```
1<NL>
```

**Example**

This example places the *PSC flag into the integer variable Pscflag.

```
10 OUTPUT 707;"*PSC?"
20 ENTER 707;Pscflag
30 PRINT Pscflag
40 END
```

# *RCL (Recall)

**Command**

```
*RCL <register>
```

The *RCL command restores the state of the oscilloscope to a setup previously stored in the specified save/recall register. An oscilloscope setup must have been stored previously in the specified register. Registers 0 through 9 are general-purpose registers and can be used by the *RCL command.

`<register>` An integer, 0 through 9, specifying the save/recall register that contains the oscilloscope setup you want to recall.

**Example**

This example restores the oscilloscope to the oscilloscope setup stored in register 3.

```
10  OUTPUT 707;"*RCL 3"
20  END
```

**See Also**

*SAV (Save). An error message appears on the oscilloscope's display if nothing has been previously saved in the specified register.

# *RST (Reset)

Command

```
*RST
```

The *RST command places the oscilloscope in a known state. This is the same as using the front-panel default setup button.

Default setup does change the the :SYSTem:HEADer or the :SYSTem:LONGform settings but does change the completion criteria (:ACQuire:COMPlete) to 90%.

**Example**

This example resets the oscilloscope to a known state.

```
10  OUTPUT 707;"*RST"
20  END
```

The default values for all of the Infiniium controls is located in the Infiniium Help System under Default Setup.

# *SAV (Save)

Command `*SAV <register>`

The *SAV command stores the current state of the oscilloscope in a save register.

`<register>` An integer, 0 through 9, specifying the register used to save the current oscilloscope setup.

**Example**

This example stores the current oscilloscope setup to register 3.

```
10  OUTPUT 707;"*SAV 3"
20  END
```

**See Also**

*RCL (Recall).

# *SRE (Service Request Enable)

Command

```
*SRE <mask>
```

The *SRE command sets the Service Request Enable Register bits. By setting the *SRE, when the event happens, you have enabled the oscilloscope's interrupt capability. The oscilloscope will then do an SRQ (service request), which is an interrupt.

`<mask>` An integer, 0 to 255, representing a mask value for the bits to be enabled in the Service Request Enable Register as shown in Table 7-4.

**Example**

This example enables a service request to be generated when a message is available in the output queue. When a message is available, the MAV bit is high.

```
10  OUTPUT 707;"*SRE 16"
20  END
```

Query

```
*SRE?
```

The *SRE? query returns the current contents of the Service Request Enable Register.

Returned Format

```
<mask><NL>
```

`<mask>` An integer, 0 to 255, representing a mask value for the bits enabled in the Service Request Enable Register.

**Example**

This example places the current contents of the Service Request Enable Register in the numeric variable, Value, then prints the value of the variable to the computer's screen.

```
10  OUTPUT 707;"*SRE?"
20  ENTER 707;Value
30  PRINT Value
40  END
```

The Service Request Enable Register contains a mask value for the bits to be enabled in the Status Byte Register. A "1" in the Service Request Enable Register enables the corresponding bit in the Status Byte Register. A "0" disables the bit.

**Table 7-4 Service Request Enable Register Bits**

| Bit | Weight | Enables |
|---|---|---|
| 7 | 128 | OPER - Operation Status Register |
| 6 | 64 | Not Used |
| 5 | 32 | ESB - Event Status Bit |
| 4 | 16 | MAV - Message Available |
| 3 | 8 | Not Used |
| 2 | 4 | MSG - Message |
| 1 | 2 | USR - User Event Register |
| 0 | 1 | TRG - Trigger |

# *STB? (Status Byte)

**Query**

```
*STB?
```

The *STB? query returns the current contents of the Status Byte, including the Master Summary Status (MSS) bit. See Table 7-5 for Status Byte Register bit definitions.

**Returned Format**

```
<value><NL>
```

`<value>` An integer, 0 to 255, representing a mask value for the bits enabled in the Status Byte.

**Example**

This example reads the contents of the Status Byte into the numeric variable, Value, then prints the value of the variable to the computer's screen.

```
10  OUTPUT 707;"*STB?"
20  ENTER 707;Value
30  PRINT Value
40  END
```

In response to a serial poll (SPOLL), Request Service (RQS) is reported on bit 6 of the status byte. Otherwise, the Master Summary Status bit (MSS) is reported on bit 6. MSS is the inclusive OR of the bitwise combination, excluding bit 6, of the Status Byte Register and the Service Request Enable Register. The MSS message indicates that the oscilloscope is requesting service (SRQ).

**Table 7-5** **Status Byte Register Bits**

| Bit | Bit Weight | Bit Name | Condition |
|---|---|---|---|
| 7 | 128 | OPER | 0 = no enabled operation status conditions have occurred<br>1 = an enabled operation status condition has occurred |
| 6 | 64 | RQS/MSS | 0 = oscilloscope has no reason for service<br>1 = oscilloscope is requesting service |
| 5 | 32 | ESB | 0 = no event status conditions have occurred<br>1 = an enabled event status condition has occurred |
| 4 | 16 | MAV | 0 = no output messages are ready<br>1 = an output message is ready |
| 3 | 8 | --- | 0 = not used |
| 2 | 4 | MSG | 0 = no message has been displayed<br>1 = message has been displayed |
| 1 | 2 | USR | 0 = no enabled user event conditions have occurred<br>1 = an enabled user event condition has occurred |
| 0 | 1 | TRG | 0 = no trigger has occurred<br>1 = a trigger occurred |
| | 0 = False = Low | | 1 = True = High |

# *TRG (Trigger)

Command

```
*TRG
```

The *TRG command has the same effect as the Group Execute Trigger message (GET) or RUN command. It acquires data for the active waveform display, if the trigger conditions are met, according to the current settings.

**Example**

This example starts the data acquisition for the active waveform display according to the current settings.

```
10  OUTPUT 707;"*TRG"
20  END
```

**Trigger Conditions Must Be Met**

When you send the *TRG command in Single trigger mode, the trigger conditions must be met before the oscilloscope will acquire data.

# *TST? (Test)

Query

```
*TST?
```

The *TST? query causes the oscilloscope to perform a self-test, and places a response in the output queue indicating whether or not the self-test completed without any detected errors. Use the :SYSTem:ERRor command to check for errors. A zero indicates that the test passed and a non-zero indicates the self-test failed.

**Disconnect Inputs First**

You must disconnect all front-panel inputs before sending the *TST? command.

Returned Format

```
<result><NL>
```

`<result>` 0 for pass; non-zero for fail.

**Example**

This example performs a self-test on the oscilloscope and places the results in the numeric variable, Results. The program then prints the results to the computer's screen.

```
10  OUTPUT 707;"*TST?"
20  ENTER 707;Results
30  PRINT Results
40  END
```

If a test fails, refer to the troubleshooting section of the service guide.

**Expanded Error Reporting**

The :SELFtest:SCOPETEST command has expanded error reporting. Instead of using *TST?, Agilent recommends that you use the :SELFtest:SCOPETEST command. In either case, be sure you disconnect all front-panel inputs before sending the *TST? command.

The self-test takes approximately 3 minutes to complete. When using timeouts in your program, a 200-second duration is recommended.

# *WAI (Wait)

Command

```
*WAI
```

The *WAI command has no function in the oscilloscope, but is parsed for compatibility with other instruments.

**Example**

```
Output 707;"*WAI"
```

8

# Root Level Commands

# Root Level Commands

Root level commands control many of the basic operations of the oscilloscope that you can select by pressing the labeled keys on the front panel. These commands are always recognized by the parser if they are prefixed with a colon, regardless of the current tree position. After executing a root level command, the parser is positioned at the root of the command tree.

These root level commands and queries are implemented in the Infiniium Oscilloscopes:

- AER? (Arm Event Register)
- AUToscale
- BLANk
- CDISplay
- DIGitize
- MTEE (Mask Test Enable Register)
- MTER? (Mask Test Event Register)
- MODel?
- OPEE (Operation Status Enable)
- OPER? (Operation Status Register)
- OVLEnable (for 54845A and 54835A only)
- OVLRegister (for 54845A and 54835A only)
- PRINt
- RECall:SETup
- RUN
- SERial (Serial Number)
- SINGle
- STOP
- STORe:SETup
- STORe:WAVeform
- TER? (Trigger Event Register)
- VIEW

# AER? (Arm Event Register)

Query

```
:AER?
```

The :AER? query reads the Arm Event Register and returns 1 or 0. After the Arm Event Register is read, the register is cleared. The returned value 1 indicates a trigger armed event has occurred and 0 indicates a trigger armed has not occurred.

**Arm Event Returns**

:AER? will allow the Arm Event to return either immediately (if you have armed but not triggered) or on the next arm (if you have already triggered). However, *CLS is always required to get an SRQ again.

Once the AER bit is set, it is cleared only by doing :AER? or by sending a *CLS command.

Returned Format

```
[:AER] {1 | 0}
```

# AUToscale

Command

```
:AUToscale
```

The :AUToscale command causes the oscilloscope to evaluate all input waveforms and find the optimum conditions for displaying the waveform. It searches each of the channels for input waveforms and shuts off channels where no waveform is found. It adjusts the vertical gain and offset for each channel that has a waveform, and sets the time base on the lowest numbered input channel that has a waveform.

The trigger is found by first searching external trigger inputs, then searching each channel, starting with channel 4, then channel 3, channel 2, and channel 1, until a trigger waveform is detected. If waveforms cannot be found on any vertical input, the oscilloscope is returned to its former state.

Autoscale sets the following:

- Channel Display, Scale, and Offset
- Trigger Sweep, Mode, Edge, Source, Level, Slope, Hysteresis, and Holdoff
- Acquisition Sampling Rate and Memory Depth
- Time Base Scale and Position
- Marker Mode Set to Measurement
- Resets Acquisition Completion Criteria to 100%

Autoscale turns off the following:

- Measurements on sources that are turned off
- Functions
- Windows
- Memories

No other controls are affected by Autoscale.

**Example**

This example automatically scales the oscilloscope for the input waveform.

```
10  OUTPUT 707;":AUTOSCALE"
20  END
```

# BLANk

Command

```
:BLANk {CHANnel<N> | FUNCtion<N> | WMEMory<N>}
```

The :BLANk command turns off an active channel, function, or waveform memory. The :VIEW command turns them on.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**Example**

This example turns off channel 1.

```
10  OUTPUT 707;":BLANK CHANNEL1"
20  END
```

# CDISplay

Command

```
:CDISplay
```

The :CDISplay command clears the display and resets all associated measurements. If the oscilloscope is stopped, all currently displayed data is erased. If the oscilloscope is running, all of the data in active channels and functions is erased; however, new data is displayed on the next acquisition. Waveform memories are not erased.

**Example**

This example clears the oscilloscope display.

```
10  OUTPUT 707;":CDISPLAY"
20  END
```

# DIGitize

**Command**

```
:DIGitize [CHANnel<N> | FUNCtion<N>][,...]
```

<N> An integer, 1 - 2, for 54810A and 54820A Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

The :DIGitize command invokes a special mode of data acquisition that is more efficient than using the :RUN command. This command initializes the selected channels or functions, then acquires them according to the current oscilloscope settings. When all waveforms are completely acquired, the oscilloscope is stopped. The waveform completion criteria is set with the ":ACQuire:COMPlete" command.

If you specify channel or function parameters, then these are the only waveforms acquired and the display waveforms of the specified channels and functions are turned off.

**Full Range of Measurement and Math Operators are Available**

Even though digitized waveforms are not displayed, you may perform the full range of measurement and math operators on them.

If you use the :DIGitize command with no parameters, the digitize operation is performed on the channels or functions that are being displayed in the Infiniium waveform viewing area. In this case, the display state of the acquired waveforms is not changed after the the :DIGitize command is completed. Because the command executes more quickly without parameters, this form of the command is useful for repetitive measurement sequences. You can also use this mode if you want to view the digitize results because the display state of the digitized waveforms is not affected.

See the Sample Programs in chapter 6 for examples of how to use :DIGitize and its related commands.

**Example**

This example acquires data on channel 1 and function 2.

```
10  OUTPUT 707;":DIGITIZE CHANNEL1,FUNCTION2"
20  END
```

The ACQuire subsystem commands set up conditions such as COUNt for the next :DIGitize command. The WAVeform subsystem commands determine how the data is transferred out of the oscilloscope, and how to interpret the data.

# MTEE

Command

```
:MTEE <enable_mask>
```

The :MTEE command is used to set bits in the Mask Test Enable Register. This register enables the following bits of the Mask Test Event Register:

<enable_mask> Bit 0 - Mask Test Complete
Bit 1 - Mask Test Fail
Bit 2 - Mask Low Amplitude
Bit 3 - Mask High Amplitude
Bit 4 - Mask Align Complete
Bit 5 - Mask Align Fail
Bit 6-7 are not used and are set to zero (0).

Query

```
:MTEE?
```

The :MTEE? query returns the value stored in the Mask Test Enable Register.

Returned Format

```
[:MTEE] <enable_mask>
```

**Example**

Suppose your application requires an interrupt whenever a Mask Test Fail occurs in the mask test register. You can enable this bit to generate the summary bit by sending:

```
OUTPUT 707;"MTEE 2"
```

Whenever an error occurs, the oscilloscope sets the MASK bit in the Operation Status Register. Because the bits in the Operation Status Enable Register are all enabled, a summary bit is generated to set bit 7 (OPER) in the Status Byte Register.

If bit 7 (OPER) in the Status Byte Register is enabled (via the *SRE command), a service request interrupt (SRQ) is sent to the external computer.

# MTER?

Query `:MTER?`

The :MTER? query returns the value stored in the Mask Test Event Register. The bits stored in the register have the following meanings:

Bit 0 Mask Test Complete bit is set whenever the mask test is complete.

Bit 1 Mask Test Fail bit is set whenever the mask test failed.

Bit 2 Mask Low Amplitude bit is set whenever the signal is below the mask amplitude.

Bit 3 Mask High Amplitude bit is set whenever the signal is above the mask amplitude.

Bit 4 Mask Align Complete bit is set whenever the mask align is complete.

Bit 5 Mask Align Fail bit is set whenever the mask align failed.

The Mask Test Event Register is read and cleared by the MTER? query. The register output is enabled or disabled using the mask value supplied with the MTEE command.

Returned Format `0-63 decimal value.`

**Disabled Mask Test Event Register Bits Respond, but Do Not Generate a Summary Bit**

Mask Test Event Register bits that are not enabled still respond to their corresponding conditions (that is, they are set if the corresponding event occurs). However, because they are not enabled, they do not generate a summary bit in the Operation Status Register.

# MODel?

Query

```
:MODel? [FRAMe]
```

The :MODel? query returns the model number for the oscilloscope frame.

Returned Format

A six-character alphanumeric model number in quotation marks. Output is determined by header and longform status as in Table 8-1.

**Table 8-1** **MODel? Returned Format**

| HEADER | | LONGFORM | | RESPONSE |
|---|---|---|---|---|
| ON | OFF | ON | OFF | |
| | X | | X | 548xx |
| | X | X | | 548xx |
| X | | | X | :MOD 548xx |
| X | | X | | :MODEL 548xx |

Where "xx" in the Response 548xx = 10A, 15A, 20A, 25A, and 45A.

**Example**

This example places the model number of the frame in a string variable, Model$, then prints the contents of the variable on the computer's screen.

```
10 Dim Model$[13]!Dimension variable
20 OUTPUT 707;":MODEL? FRAME"
30 ENTER 707; Model$
40 PRINT MODEL$
50 END
```

# OPEE

**Command** `:OPEE <mask>`

`<mask>` The decimal weight of the enabled bits.

The :OPEE command sets a mask in the Operation Status Enable register. Each bit that is set to a "1" enables that bit to set bit 7 in the status byte register, and potentially causes an SRQ to be generated. Bit 5, Wait for Trig is used. Other bits are reserved.

**Query** `:OPEE?`

The query returns the current value contained in the Operation Status Enable register as a decimal number.

**Returned Format** `[OPEE] <value><NL>`

# OPER?

Query

```
:OPER?
```

The :OPER? query returns the value contained in the Operation Status Register as a decimal number. This register hosts the WAIT TRIG bit (bit 5) and the PROG bit (bit 14).

The WAIT TRIG bit is set by the Trigger Armed Event Register and indicates that the trigger is armed. The PROG bit is reserved for future use.

Returned Format

```
[OPER] <value><NL>
```

# OVLEnable

**Command**

```
:OVLEnable <enable_mask>
```

The :OVLEnable command enables the built-in overload protection in the 54845A and 54835A oscilloscope. If too much voltage is present at the channel input, the oscilloscope turns the channel off. When the voltage load is removed, the channel returns automatically. This command is available only on the 54845A and 54835A.

`<enable_mask>` The overload enable mask is an integer representing a channel as follows:

Bit 0 - Channel 1
Bit 1 - Channel 2
Bit 2 - Channel 3
Bit 3 - Channel 4
Bits 7-4 are not used.

**Query**

```
:OVLEnable?
```

The :OVLEnable? query returns the current value contained in the Overload Enable Register.

**Returned Format**

```
[OVLEnable] <enable_mask><NL>
```

**See Also**

:CHANnel<N>:PROTection:CLEar

## OVLRegister?

Query

```
:OVLRegister?
```

The :OVLRegister? query returns the overload value stored in the overload register. This query is available only on the 54845A and 54835A.

Returned Format

```
[OVLRegister] <value><NL>
```

# PRINt

Command

```
:PRINt
```

The :PRINt command outputs a copy of the screen to a printer or other device destination specified in the HARDcopy subsystem. You can specify the selection of the output and the printer using the HARDcopy subsystem commands.

**Example**

This example outputs a copy of the screen to a printer or a disk file.

```
10  OUTPUT 707;":PRINT"
20  END
```

# RECall:SETup

**Command**

```
:RECall:SETup <setup_memory_num>
```

`<setup_memory_num>` Setup memory number, an integer, 0 through 9.

The :RECall:SETup command recalls a setup that was saved in one of the oscilloscope's setup memories. You can save setups using either the :STORe:SETup command or the front panel.

**Examples**

This command recalls a setup from setup memory 2.

```
10 OUTPUT 707;":RECall:SETup 2"
20 END
```

# RUN

Command

```
:RUN
```

The :RUN command starts the oscilloscope running. When the oscilloscope is running, it acquires waveform data according to its current settings. Acquisition runs repetitively until the oscilloscope receives a :STOP command, or until there is only one acquisition if Trigger Sweep is set to Single.

**Example**

This example causes the oscilloscope to acquire data repetitively.

```
10  OUTPUT 707;":RUN"
20  END
```

# SERial (Serial Number)

Command

```
:SERial {[FRAMe],<serial_number>}
```

<serial_number> A ten-character alphanumeric serial number enclosed with quotation marks.

The :SERial command sets the serial number for the oscilloscope frame. The serial number is entered by Agilent Technologies. Therefore, setting the serial number is not normally required unless the oscilloscope is serialized for a different application.

The oscilloscope's serial number is part of the string returned for the *IDN? query described in the Common Commands chapter.

**Example**

This example sets the serial number for the oscilloscope's frame to "US12345678".

```
10  OUTPUT 707;":SERIAL FRAME,""US12345678"""
20  END
```

Query

```
:SERial? [FRAMe]
```

The query returns the current serial number string for the specified frame.

Returned Format

```
[:SERial FRAMe] US12345678
```

**Example**

This example places the serial number for the oscilloscope frame in the string variable Serial?, then prints the contents of the variable to the computer's screen.

```
10 Dim Serial$[50]!Dimension variable
20 OUTPUT 707;":SERIAL? FRAME"
30 ENTER 707; Serial$
40 PRINT SERIAL$
50 END
```

# SINGle

Command

```
:SINGle
```

The :SINGle command causes the oscilloscope to make a single acquisition when the next trigger event occurs.

**Example**

This example sets up the oscilloscope to make a single acquisition when the next trigger event occurs.

```
10  OUTPUT 707;":SINGLE"
20  END
```

**See Also**

:TRIGger:SWEep AUTO|TRIGgered|SINGle for how to turn the single sweep off.

# STOP

Command

```
:STOP
```

The :STOP command causes the oscilloscope to stop acquiring data. To restart the acquisition, use the :RUN or :SINGle command.

**Example**

This example stops the current data acquisition.

```
10  OUTPUT 707;":STOP"
20  END
```

# STORe:SETup

**Command**

```
:STORe:SETup <setup_memory_num>
```

`<setup_memory_num>` Setup memory number, an integer, 0 through 9.

The :STORe:SETup command saves the current oscilloscope setup in one of the setup memories.

**Example**

This example stores the current oscilloscope setup to setup memory 0.

```
10  OUTPUT 707;":STORE:SETUP 0"
20  END
```

# STORe:WAVeform

Command

```
:STORe:WAVeform {{CHANnel<N> | FUNCtion<N> |
WMEMory<N>},{WMEMory<N>}}
```

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

The :STORe:WAVeform command copies a channel, function, or stored waveform to a waveform memory. The parameter preceding the comma specifies the source and can be any channel, function, or waveform memory. The parameter following the comma is the destination, and can be any waveform memory.

**Example**

This example copies channel 1 to waveform memory 3.

```
10  OUTPUT 707;":STORE:WAVEFORM CHANNEL1,WMEMORY3"
20  END
```

# TER? (Trigger Event Register)

Query

```
:TER?
```

The :TER? query reads the Trigger Event Register. A "1" is returned if a trigger has occurred. A "0" is returned if a trigger has not occurred.

Returned Format

```
{1 | 0}<NL>
```

**Example**

This example checks the current status of the Trigger Event Register, places the status in the string variable, Current$, then prints the contents of the variable to the computer's screen.

```
10  DIM Current$[50]!Dimension variable
20  OUTPUT 707;":TER?"
30  ENTER 707;Current$
40  PRINT Current$
50  END
```

Once this bit is set, you can clear it only by reading the register with the :TER? query, or by sending a *CLS common command. After the Trigger Event Register is read, it is cleared.

# VIEW

Command

```
:VIEW {CHANnel<N> | FUNCtion<N> | WMEMory<N>}
```

The :VIEW command turns on a channel, function, or waveform memory.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**Example**

This example turns on channel 1.

```
10  OUTPUT 707;":VIEW CHANNEL1"
20  END
```

**See Also**

The :BLANk command turns off a channel, function, or waveform memory.

9

# System Commands

# System Commands

SYSTem subsystem commands control the way query responses are formatted, send and receive setup strings, and enable reading and writing to the advisory line of the oscilloscope. You can also set and read the date and time in the oscilloscope using the SYSTem subsystem commands.

These SYSTem commands and queries are implemented in the Infiniium Oscilloscopes:

- DATE
- DEBug
- DSP
- ERRor?
- HEADer
- HELP:HEADers?
- LANGuage
- LONGform
- SETup
- TIME

# DATE

**Command**

```
:SYSTem:DATE <day>,<month>,<year>
```

The :SYSTem:DATE command sets the date in the oscilloscope, and is not affected by the *RST common command.

`<year>` Specifies the year in the format <yyyy> | <yy>. The values range from 1992 to 2035.

`<month>` Specifies the month in the format <1, 2, . . . 12> | <JAN, FEB, MAR . . .>.

`<day>` Specifies the day in the format <1 . . . 31>.

**Example**

This example sets the date to July 1, 1997.

```
10  OUTPUT 707;":SYSTEM:DATE 1,7,97"
20  END
```

**Query**

```
:SYSTem:DATE?
```

The :SYSTem:DATE? query returns the current date in the oscilloscope.

**Returned Format**

```
[:SYSTem:DATE] <day> <month> <year><NL>
```

**Example**

This example queries the date.

```
10 DIM Date$ [50]
20 OUTPUT 707;":SYSTEM:DATE?"
30 ENTER 707; Date$
40 PRINT Date$
```

# DEBug

Command

```
:SYSTem:DEBug {{ON|1}[,<output_mode>[,"<file_name>"
[,<create_mode>]]] | {OFF|0}}
```

The :SYSTem:DEBug command turns the debug/trace m ode on and off. This mode enables the tracing of incoming GPIB commands. If you select CREate mode, a new file is created, and/or an existing file is overwritten. If you select APPend mode, the information is appended to an existing file. The :SYSTem:DEBug command shows any header and/or parameter errors.

The default create mode is CREate, the default output mode is FileSCReen, and the default file name is c:\scope\data\debug.txt. In debug mode, the File View button lets you view the current debug file, or any other debug file. This is a read-only mode.

`<output_mode>` {FILE | SCReen | FileSCReen}

`<file_name>` An MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used). The file name assumes the present working directory if a path does not precede the file name.

`<create_mode>` {CREate | APPend}

**Examples**

This example turns on the debug/trace mode and creates a debug file.

```
10  OUTPUT 707;":SYSTEM:DEBUG ON,FILE,
    "C:\scope\data\pacq8xx.txt",CREATE"
20  END
```

The created file resembles:

```
Debug information file C:\scope\data\pacq8xx.txt
Date:  14 NOV 1997
Time:  09:59:35
Model: 54815A
Serial#:  sn ?
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$
>:ACQuire:BEST FLATness$<NL>
?     ^
?-113, Undefined header
>:syst:err? string$<NL>
<:SYSTEM:ERROR -113,"Undefined header"$
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$
```

This example appends information to the debug file.

```
10  OUTPUT 707;":SYSTEM:DEBUG ON,FILE,
    "C:\scope\data\pacq8xx.txt",APPEND"
20  END
```

After appending information, the file resembles:

```
Debug information file C:\scope\data\pacq8xx.txt
Date:  14 NOV 1997
Time:  09:59:35
Model: 54815A
Serial#:  sn ?
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$
>:ACQuire:BEST FLATness$<NL>
?     ^
?-113, Undefined header
>:syst:err? string$<NL>
<:SYSTEM:ERROR -113,"Undefined header"$
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$

Debug information file C:\scope\data\pacq8xx.txt appended
Date:  14 NOV 1997
Time:  10:10:35
Model: 54815A
Serial#:   sn ?
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$
>:ACQuire:BEST FLATness$<NL>
?     ^
?-113, Undefined header
>:syst:err? string$<NL>
<:SYSTEM:ERROR -113,"Undefined header"$
```

---

Query

```
:SYSTem:DEBug?
```

The :SYSTem:DEBug? query returns the current debug mode settings.

Returned Format

```
[:SYSTem:DEBug]  {{1,<output_mode>,"<file_name>",
<create_mode>} | 0} <NL>
```

# DSP

Command

```
:SYSTem:DSP "<string>"
```

The :SYSTem:DSP command writes a quoted string, excluding quotation marks, to the advisory line of the instrument display. If you want to clear a message on the advisory line, send a null (empty) string.

`<string>` An alphanumeric character array up to 89 bytes long.

**Example**

This example writes the message, "Test 1" to the advisory line of the oscilloscope.

```
10  OUTPUT 707;":SYSTEM:DSP ""Test 1"""
20  END
```

Query

```
:SYSTem:DSP?
```

The :SYSTem:DSP? query returns the last string written to the advisory line. This may be a string written with a :SYSTem:DSP command, or an internally generated advisory.

The string is actually read from the message queue. The message queue is cleared when it is read. Therefore, the displayed message can only be read once over the bus.

Returned Format

```
[:SYSTem:DSP] <string><NL>
```

**Example**

This example places the last string written to the advisory line of the oscilloscope in the string variable, Advisory$. Then, it prints the contents of the variable to the computer's screen.

```
10  DIM Advisory$[89]!Dimension variable
20  OUTPUT 707;":SYSTEM:DSP?"
30  ENTER 707;Advisory$
40  PRINT Advisory$
50  END
```

# ERRor?

**Query**

```
:SYSTem:ERRor? [{NUMBer | STRing}]
```

The :SYSTem:ERRor? query outputs the next error number in the error queue over the GPIB. When either NUMBer or no parameter is specified in the query, only the numeric error code is output. When STRing is specified, the error number is output followed by a comma and a quoted string describing the error. Table 29-1 lists the error numbers and their corresponding error messages.

**Returned Format**

```
[:SYSTem:ERRor] <error_number>[,<quoted_string>]<NL>
```

`<error_number>` A numeric error code.

`<quoted_string>` A quoted string describing the error.

**Example**

This example reads the oldest error number and message in the error queue into the string variable, Condition$, then prints the contents of the variable to the computer's screen.

```
10  DIM Condition$[64]!Dimension variable
20  OUTPUT 707;":SYSTEM:ERROR? STRING"
30  ENTER 707;Condition$
40  PRINT Condition$
50  END
```

Infiniium Oscilloscopes have an error queue that is 30 errors deep and operates on a first-in, first-out (FIFO) basis. Successively sending the :SYSTem:ERRor? query returns the error numbers in the order that they occurred until the queue is empty. When the queue is empty, this query returns headers of 0, "No error." Any further queries return zeros until another error occurs. Note that front-panel generated errors are also inserted in the error queue and the Event Status Register.

**Send *CLS Before Other Commands or Queries**

Send the *CLS common command to clear the error queue and Event Status Register before you send any other commands or queries.

**See Also**

The "Error Messages" chapter for more information on error messages and their possible causes.

# HEADer

Command

```
:SYSTem:HEADer {{ON|1} | {OFF|0}}
```

The :SYSTem:HEADer command specifies whether the instrument will output a header for query responses. When :SYSTem:HEADer is set to ON, the query responses include the command header.

**Example**

This example sets up the oscilloscope to output command headers with query responses.

```
10  OUTPUT 707;":SYSTEM:HEADER ON"
20  END
```

Query

```
:SYSTem:HEADer?
```

The :SYSTem:HEADer? query returns the state of the :SYSTem:HEADer command.

Returned Format

```
[:SYSTem:HEADer] {1|0}<NL>
```

**Example**

This example examines the header to determine the size of the learn string. Memory is then allocated to hold the learn string before reading it. To output the learn string, the header is sent, then the learn string and the EOF.

```
10  DIM Header$[64]
20  OUTPUT 707;"syst:head on"
30  OUTPUT 707;":syst:set?"
40  More_chars:  !
50  ENTER 707 USING "#,A";This_char$
60  Header$=Header$&This_char$
70  IF This_char$<>"#" THEN More_chars
80  !
90  ENTER 707 USING "#,D";Num_of_digits
100 ENTER 707 USING "#,"&VAL$(Num_of_digits)&"D";Set_size
110 Header$=Header$&"#"&VAL$(Num_of_digits)&VAL$(Set_size)
120 !
130 ALLOCATE INTEGER Setup(1:Set_size)
140 ENTER 707 USING "#,B";Setup(*)
150 ENTER 707 USING "#,A";Eof$
160 !
170 OUTPUT 707 USING "#,-K";Header$
180 OUTPUT 707 USING "#,B";Setup(*)
190 OUTPUT 707 USING "#,A";Eof$
200 END
```

**Turn Headers Off when Returning Values to Numeric Variables**

Turn headers off when returning values to numeric variables. Headers are always off for all common command queries because headers are not defined in the IEEE 488.2 standard.

# HELP:HEADers

Query

```
:SYSTem:HELP:HEADers? [<language>]
```

The :SYSTem:HELP:HEADers? query returns a list of all commands and queries for the selected language. The default language is HP548XX.

`<language>` {HP548XX | HP547XX | HP545XX}

**Example**

This example generates three files that contain all of the commands and queries for the three different languages.

```
10  !  do headers transfer
20    DIM Header$[64],A$[8192],Headerfile$[128]
30    S=707
40    OUTPUT S;"*cls"
50    OUTPUT S;":syst:head on;long on "
60    !
70    Show_all=0
80    !
90    FOR I=1 TO 3
100   IF I=1 THEN Language$="HP545XX"
110   IF I=2 THEN Language$="HP547XX"
120   IF I=3 THEN Language$="HP548XX"
130   OUTPUT S;":syst:help:headers? "&Language$
140   Header$=""
150   !
160 More_chars:  !
170   ENTER S USING "#,A";This_char$
180   IF This_char$<>"#" THEN Header$=Header$&This_char$
190   IF This_char$<>"#" THEN More_chars
200   !
210   ENTER S USING "#,D";Num_of_digits
220   ENTER S USING "#,"&VAL$(Num_of_digits)&"D";Set_size
230   Header$=Header$&"#"&VAL$(Num_of_digits)&VAL$(Set_size)
240   !
250   PRINT Header$,Set_size
260   !
270   Headerfile$="H"&Language$
280   Eof$=CHR$(10)
290   ON ERROR GOTO Off_error
300   PURGE Headerfile$
310   !
320 Off_error:  !
330   OFF ERROR
340   !
```

```
350   CREATE Headerfile$,100
360   ASSIGN @Headerfile TO Headerfile$;FORMAT ON
370   !
375   OUTPUT @Headerfile;CHR$(9)&Language$
380   !
390   ON TIMEOUT 7,5 GOTO Lines_done
400 More_lines:   !
410   ENTER S;A$
420   IF Show_all=1 THEN PRINT A$
430   A$=CHR$(9)&A$
440   OUTPUT @Headerfile;A$
450   GOTO More_lines
460   !
470 Lines_done:   !
480   ASSIGN @Headerfile TO *
490   OFF TIMEOUT
500   !
510   NEXT I
520   STOP
530 Get_ans:   !
540   ENTER S;A$
550   PRINT A$
560   RETURN
570   !
580   END
```

**Returned Format** `[:SYSTem:HELP:HEADers] #nd..d<definite_block_data><NL>`

`<n>` Number of digits to follow.

`<d..d>` Size of definite block data.

**Example** This example shows a query return with HEADer on and LONGform off.

```
:SYST:HELP:HEAD #517734<NL>
:ACOMplete?<NL>
:ACQuire:AVERage<NL>
:ACQuire:AVERage:COUNt<NL>
:ACQuire:BWLimit<NL>
*TRG<NL>
*TST?<NL>
*WAI<NL>
```

# LANGuage

Command

```
:SYSTem:LANGuage {HP548XX | HP547XX | HP545XX}
```

The :SYSTem:LANGuage command selects the programming language for the Infiniium Oscilloscope. At powerup, the default language is HP548XX. The programming language does not change when either *RST or AUToscale are sent over the bus.

**Example**

This example sets the programming language for the Infiniium Oscilloscope to the language used in 545xx Oscilloscopes.

```
10  OUTPUT 707;":SYSTEM:LANGUAGE HP545XX"
20  END
```

Query

```
:SYSTem:LANGuage?
```

The :SYSTem:LANGuage? query returns the currently selected programming language.

Returned Format

```
[:SYSTem:LANGuage] <selected_language><NL>
```

# LONGform

Command

```
:SYSTem:LONGform {{ON|1} | {OFF|0}}
```

The :SYSTem:LONGform command specifies the format for query responses. If the LONGform is set to OFF, command headers and alpha arguments are sent from the oscilloscope in the short form (abbreviated spelling). If LONGform is set to ON, the whole word is output.

**Example**

This example sets the format for query responses from the oscilloscope to the short form (abbreviated spelling).

```
10  OUTPUT 707;":SYSTEM:LONGFORM OFF"
20  END
```

Query

```
:SYSTem:LONGform?
```

The :SYSTem:LONGform? query returns the current state of the :SYSTem:LONGform command.

Returned Format

```
[:SYSTem:LONGform] {1|0}<NL>
```

**Example**

This example checks the current format for query responses from the oscilloscope, and places the result in the string variable, Result$. Then, it prints the contents of the variable to the computer's screen.

```
10  DIM Result$[50]!Dimension variable
20  OUTPUT 707;":SYSTEM:LONGFORM?"
30  ENTER 707;Result$
40  PRINT Result$
50  END
```

**LONGform Does Not Affect Input Headers and Arguments**

LONGform has no effect on input headers and arguments sent to the instrument. You may send headers and arguments to the oscilloscope in either the long form or short form, regardless of the current state of the :SYSTem:LONGform command.

# SETup

Command

```
:SYSTem:SETup <binary_block_data>
```

The :SYSTem:SETup command sets up the oscilloscope as defined by the data in the setup string from the computer.

`<binary_block_data>` A string, consisting of bytes of setup data. The number of bytes is a dynamic number that is read and allocated by oscilloscope's software.

**Example**

This example sets up the instrument as defined by the setup string stored in the variable, Set$.

```
10  OUTPUT 707 USING "#,-K";":SYSTEM:SETUP ";Set$
20  END
```

**HP BASIC Image Specifiers**

# is an HP BASIC image specifier that suppresses the automatic output of the EOI sequence following the last output item.

K is an HP BASIC image specifier that outputs a number or string in standard form with no leading or trailing blanks.

Query

```
:SYSTem:SETup?
```

The :SYSTem:SETup? query outputs the oscilloscope's current setup to the computer in binary block data format as defined in the IEEE 488.2 standard.

Returned Format

```
[:SYSTem:SETup] #NX...X<setup_data_string><NL>
```

The first character in the setup data string is a number added for disk operations.

**Example**

This example stores the current oscilloscope setup in the string variable, Set$.

```
10  DIM Set$[15000]!Dimension variable
20  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
30  OUTPUT 707;":SYSTEM:SETUP?"
40  ENTER 707 USING "-K";Set$
50  END
```

**HP BASIC Image Specifiers**

–K is an HP BASIC image specifier which places the block data in a string, including carriage returns and line feeds, until EOI is true, or the dimensioned length of the string is reached.

**:SYSTem:SETup Can Operate Just Like *LRN?**

When headers and LONGform are on, the :SYSTem:SETup? query operates the same as the *LRN? query in the common commands. Otherwise, *LRN? and :SYSTem:SETup are not interchangeable.

# TIME

**Command**

```
:SYSTem:TIME <hour>,<minute>,<second>
```

The :SYSTem:TIME command sets the time in the oscilloscope to 10:30:45, and is not affected by the *RST common command.

`<hour>` 0..23

`<minute>` 0..59

`<second>` 0..59

**Example**

This example sets the oscilloscope time to 10:30:45 p.m.

```
10  OUTPUT 707;":SYSTEM:TIME 10,30,45"
20  END
```

**Query**

```
:SYSTem:TIME?
```

The :SYSTem:TIME? query returns the current time in the oscilloscope.

**Returned Format**

```
[:SYSTem:TIME] <hour>,<minute>,<second>
```

10

# Acquire Commands

# Acquire Commands

The ACQuire subsystem commands set up conditions for executing a :DIGitize root level command to acquire waveform data. The commands in this subsystem select the type of data, the number of averages, and the number of data points.

These ACQuire commands and queries are implemented in the Infiniium Oscilloscopes:

- AllowMaxSR
- AVERage
- AVERage:COUNt
- BWLimit
- COMPlete
- COMPlete:STATe
- CONFig (for 54846A, 54845A, and 54835A only)
- INTerpolate
- MODE
- POINts (memory depth)
- POINts:AUTO
- SRATe (sampling rate)
- SRATe:AUTO

# AllowMaxSR

**Command**

```
:ACQuire:AllowMaxSR {CHANnel<N>}
```

The :ACQuire:AllowMaxSR command is used to set the 54846A, 54845A, and 54835A oscilloscopes into the two channel 8 GSa/s mode (54846A/45A) or the two channel 4GSa/s mode (54835A) when the channel number is 1 or 3. If the channel number is 2 or 4 the 54846A/45A has a maximum sample rate of 4 GSa/s and the 54835A has a maximum sample rate of 2 GSa/s.
This command is designed to be used in a mask template file.

The :ACQuire:SRATe command sets the sample rate.

> The 54810A, 54815A, 54820A, 54825A oscilloscopes accept the :ACQuire:AllowMaxSR command but do nothing.

<N> An integer, 1-4.

**Example**

This example puts an 54846A/45A into the two channel 8 GSa/s mode or an 54835A into the two channel 4 GSa/s mode.

```
10  OUTPUT 707;":ACQUIRE:ALLOWMAXSR CHANNEL 1"
20  END
```

# AVERage

Command

```
:ACQuire:AVERage {{ON|1} | {OFF|0}}
```

The :ACQuire:AVERage command enables or disables averaging. When ON, the oscilloscope acquires multiple data values for each time bucket, and averages them. When OFF, averaging is disabled. To set the number of averages, use the :ACQuire:AVERage:COUNt command described next.

Averaging is not available in PDETect mode.

The :MTESt:AVERage command performs the same function as this command.

**Example**

This example turns averaging on.

```
10  OUTPUT 707;":ACQUIRE:AVERAGE ON"
20  END
```

Query

```
:ACQuire:AVERage?
```

The :ACQuire:AVERage? query returns the current setting for averaging.

Returned Format

```
[:ACQuire:AVERAGE] {1|0}<NL>
```

**Example**

This example places the current settings for averaging into the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":ACQUIRE:AVERAGE?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# AVERage:COUNt

**Command**

```
:ACQuire:AVERage:COUNt <count_value>
```

The :ACQuire:[AVERage:]COUNt command sets the number of averages for the waveforms. In the AVERage mode, the :ACQuire:AVERage:COUNt command specifies the number of data values to be averaged for each time bucket before the acquisition is considered complete for that time bucket.

The :MTESt:AVERage:COUNt command performs the same function as this command.

`<count_value>` An integer, 2 to 4096, specifying the number of data values to be averaged.

**Example**

This example specifies that 16 data values must be averaged for each time bucket to be considered complete. The number of time buckets that must be complete for the acquisition to be considered complete is specified by the :ACQuire:COMPlete command.

```
10  OUTPUT 707;":ACQUIRE:COUNT 16"
20  END
```

**Query**

```
:ACQuire:COUNt?
```

The :ACQuire:COUNt? query returns the currently selected count value.

**Returned Format**

```
[:ACQuire:COUNt] <value><NL>
```

`<value>` An integer, 2 to 4096, specifying the number of data values to be averaged.

**Example**

This example checks the currently selected count value and places that value in the string variable, Result$. The program then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":ACQUIRE:AVERAGE:COUNT?"
30  ENTER 707;Result
40  PRINT Result
50  END
```

# BWLimit

Command

```
:ACQuire:BWLimit {{ON|1} | {OFF|0}}
```

The :ACQuire:BWLimit command controls the acquisition of filtering. The 9-bit, high-resolution bandwidth limit filter only applies to data acquired in the real time sampling mode. This command does not pertain to the equivalent time mode.

ON The digital bandwidth limit filter passes the raw data through a filter which limits the bandwidth to approximately

$$\frac{f_s}{20}$$

where $f_s$ is the current sample frequency.

OFF The filter is turned off.

**Example**

This example turns the bandwidth limit filter off.

```
10  OUTPUT 707;":ACQUIRE:BWLIMIT OFF"
20  END
```

Query

```
:ACQuire:BWLimit?
```

The :ACQuire:BWLimit? query returns the current bandwidth limit filter state.

Returned Format

```
[:ACQuire:BWLimit] {1|0}<NL>
```

**Example**

This example places the current setting of the bandwidth limit filter in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":ACQUIRE:BWLIMIT?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# COMPlete

Command

```
:ACQuire:COMPlete <percent>
```

The :ACQuire:COMPlete command specifies how many of the data point storage bins (time buckets) in the waveform record must contain a waveform sample before a measurement will be made. For example, if the command :ACQuire:COMPlete 60 has been sent, 60% of the storage bins in the waveform record must contain a waveform data sample before a measurement is made.

- If :ACQuire:AVERage is set to OFF, the oscilloscope only needs one value per time bucket for that time bucket to be considered full.
- If :ACQuire:AVERage is set to ON, each time bucket must have $n$ hits for it to be considered full, where $n$ is the value set by :ACQuire:AVERage:COUNt.

Due to the nature of real time acquisition, 100% of the waveform record bins are filled after each trigger event, and all of the previous data in the record is replaced by new data when :ACQuire:AVERage is off. Hence, the complete mode really has no effect, and the behavior of the oscilloscope is the same as when the completion criteria is set to 100% (this is the same as in PDETect mode). When :ACQuire:AVERage is on, all of the previous data in the record is replaced by new data.

The range of the :ACQuire:COMPlete command is 0 to 100 and indicates the percentage of time buckets that must be full before the acquisition is considered complete. If the complete value is set to 100%, all time buckets must contain data for the acquisition to be considered complete. If the complete value is set to 0, then one acquisition cycle will take place. Completion is set by default setup or *RST to 90%. Autoscale changes it to 100%.

`<percent>` An integer, 0 to 100, representing the percentage of storage bins (time buckets) that must be full before an acquisition is considered complete.

**Example**

This example sets the completion criteria for the next acquisition to 90%.

```
10  OUTPUT 707;":ACQUIRE:COMPLETE 90"
20  END
```

# CONFig

Command

```
:ACQuire:CONFig {TwoCHannel | FourCHannel}
```

The :ACQuire:CONFig command configures the 54846A, 54845A, and 54835A oscilloscopes to use 2 or 4 channels for acquisitions. This command is only valid for the 54846A, 54845A, and 54835A.

**Example**

This example configures the 54846A, 54845A, and 54835A oscilloscopes to use 4 channels for acquisitions.

```
10  OUTPUT 707;":ACQUIRE:CONFIG FCH"
20  END
```

Query

```
:ACQuire:CONFig?
```

The :ACQuire:CONFig? query returns the configured number of channels for acquisitions on the 54846A, 54845A, and 54835A.

Returned Format

```
[:ACQuire:CONFig] {TwoCHannel | FourCHannel}<NL>
```

## INTerpolate

**Command**

```
:ACQuire:INTerpolate {{ON|1} | {OFF|0}}
```

The :ACQuire:INTerpolate command turns the sin(x)/x interpolation filter on or off when the oscilloscope is in real time acquisition mode.

**Query**

```
:ACQuire:INTerpolate?
```

The :ACQuire:INTerpolate? query returns the current state of the sin(x)/x interpolation filter control.

**Returned Format**

```
[:ACQuire:INTerpolate] {1|0}<NL>
```

# MODE

Command

```
:ACQuire:MODE {RTIMe|{ETIMe|REPetitive}|PDETect}
```

The :ACQuire:MODE command sets the acquisition mode of the oscilloscope. Sampling mode can be Equivalent Time (Repetitive), Real Time, or Peak Detect.

`RTIMe` In Real Time mode, the complete data record is acquired on a single trigger event.

`ETIMe or REPetitive` In Equivalent Time (Repetitive) mode, the data record is acquired over multiple trigger events.

`PDETect` In Peak Detect mode, the oscilloscope acquires all of the waveform data points during one trigger event, similar to the Real Time mode; however, the rate at which data is stored to memory is limited to 250 MSa/s. Waveform anomalies are detected between samples because the oscilloscope internally acquires data at a faster speed than the selected sample rate. From these extra samples, the minimum and maximum values are determined for each sample point. The minimum and maximum values for each sample point are then displayed. This mode is not available in the 54846A, 54845A, and 54835A Oscilloscopes.

**Example**

This example sets the acquisition mode to Real Time.

```
10  OUTPUT 707;":ACQUIRE:MODE RTIME"
20  END
```

Query

```
:ACQuire:MODE?
```

The :ACQuire:MODE? query returns the current acquisition sampling mode.

Returned Format

```
[:ACQuire:MODE] {RTIMe | {ETIMe | REPetitive} | PDETect}<NL>
```

**Example**

This example places the current acquisition mode in the string variable, Mode$, then prints the contents of the variable to the computer's screen.

```
10  DIM Mode$[50]!Dimension variable
20  OUTPUT 707;":ACQUIRE:MODE?"
30  ENTER 707;Mode$
40  PRINT Mode$
50  END
```

# POINts

**Command**

```
:ACQuire:POINts {AUTO|<points_value>}
```

The :ACQuire:POINts command sets the requested memory depth for an acquisition. Before you download data from the oscilloscope to your computer, always query the points value with the :WAVeform:POINts? query or :WAVeform:PREamble? query to determine the actual number of acquired points.

You can set the points value to AUTO, which allows Infiniium to select the optimum memory depth and display update rate.

`<points_value>` An integer representing the memory depth.

The range of points available for a single channel depends on the oscilloscope model and the acquistion mode setting, as shown in Table 10-1:

**Table 10-1 Points Value Ranges**

| | 54810A/15A/ 20A/25A | 54846A/45A/35A 2-channel mode | 54846A/45A/35A 4-channel mode |
|---|---|---|---|
| **Real Time mode** | 16 to 32768 | 16 to 65536 | 16 to 32768 |
| **Equivalent Time mode** | 16 to 32768 | mode not available | 16 to 32768 |
| **Peak Detect mode** | 16 to 16384 | mode not available | mode not available |

Equivalent Time mode takes the oscilloscope out of 2 channel, 4 GSa/s (54835A) or 2 channel, 8 GSa/s (54846A/45A,).

**Example**

This example sets the memory depth to 500 points.

```
10  OUTPUT 707;":ACQUIRE:POINTS 500"
20  END
```

Query

```
:ACQuire:POINts?
```

The :ACQuire:POINts? query returns the value of the memory depth control.

Returned Format

```
[:ACQuire:POINts] <points_value><NL>
```

**Example**

This example checks the current setting for memory depth and places the result in the variable, Length. Then the program prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":ACQUIRE:POINTS?"
30  ENTER 707;Length
40  PRINT Length
50  END
```

**See Also**

:WAVeform:DATA

# POINts:AUTO

Command

```
:ACQuire:POINts:AUTO {{ON | 1} |{OFF | 0}}
```

The :ACQuire:POINts:AUTO command enables (automatic) or disables (manual) the automatic memory depth selection control. When enabled, Infiniium chooses a memory depth that optimizes the amount of waveform data and the display update rate. When disabled, you can select the amount of memory using the :ACQuire:POINts command.

**Example**

This example sets the automatic memory depth control to off.

```
10  OUTPUT 707;":ACQUIRE:POINTS:AUTO OFF"
20  END
```

Query

```
:ACQuire:POINts:AUTO?
```

The :ACQuire:POINts:AUTO? query returns the automatic memory depth control state.

Returned Format

```
[:ACQuire:POINts:AUTO] {1 | 0}<NL>
```

**Example**

This example checks the current setting for automatic memory depth control and places the result in the variable, State. Then the program prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":ACQUIRE:POINTS:AUTO?"
30  ENTER 707;State
40  PRINT State
50  END
```

**See Also**

:WAVeform:DATA

# SRATe (Sample RATe)

Command

```
:ACQuire:SRATe {AUTO | MAX | <rate>}
```

The :ACQuire:SRATe command sets the acquisition sampling rate for real time and peak detect sampling modes. If the oscilloscope is in the equivalent time sampling mode, the SRATe command has no effect on the sampling rate. However, if you change the sampling mode to real time or peak detect sampling the control will show the new value.

`AUTO` The AUTO rate allows the oscilloscope to select a sample rate that best accommodates the selected memory depth and sweep speed.

`MAX` The MAX rate enables the oscilloscope to select maximum available sample rate.

`<rate>` A real number representing the sample rate. You can send any value, but the value is rounded to the next fastest sample rate.

**Table 10-2**

**Sample Rate Minimum and Maximum Values**

| Acquisition Mode | 54810A/54815A | 54820A/54825A | 54835A 2-channel mode | 54835A 4-channel mode | 54846A/45A, 2-channel mode | 54846A/45A, 4-channel mode |
|---|---|---|---|---|---|---|
| Real Time mode | 500E-3 - 1.0E+9 | 500E-3 - 2.0E+9 | 500E-3 - 4.0E+9 | 500E-3 - 2.0E+9 | 500E-3 - 8.0E+9 | 500E-3 - 4.0E+9 |
| Peak Detect mode | 500E-3 - 250E+6 * | 500E-3 - 250E+6 * | | | | |

* Effective sample rates; internally the oscilloscope is sampling at 1 GSa/s.

**Table 10-3**

**Available Sample Rate Values (in Sa/s)**

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| .5 | 1 | 2.5 | 5 | 10 | 25 | 50 | 100 | 250 | 500 | 1K | 2.5K | 5K | 10K | 25K |
| 50K | 100K | 250K | 500K | 1M | 2.5M | 5M | 10M | 25M | 50M | 100M | 250M | 500M | 1G | 2G |
| 4G | 8G | | | | | | | | | | | | | |

The sample rate of 4 GSa/s is available on the 54835A.
The sample rates of 4 GSa/s and 8 GSa/s are available on 54846A/45A,.

**Example**

This example sets the sample rate to 250 MSa/s.

```
10  OUTPUT 707;":ACQUIRE:SRATE 250E+6"
20  END
```

Query

```
:ACQuire:SRATe?
```

The :ACQuire:SRATe? query returns the current acquisition sample rate.

Returned Format

```
[:ACQuire:SRATe] {AUTO | <rate>}<NL>
```

**Example**

This example places the current sample rate in the string variable, Sample$, then prints the contents of the variable to the computer's screen.

```
10  DIM Sample$[50]!Dimension variable
20  OUTPUT 707;":ACQUIRE:SRATE?"
30  ENTER 707;Sample$
40  PRINT Sample$
50  END
```

# SRATe:AUTO

Command

```
:ACQuire:SRATe:AUTO {{ON | 1} | {OFF | 0}}
```

The :ACQuire:SRATe:AUTO command enables or disables the automatic sampling rate selection control for real time and peak detect sampling modes. If the oscilloscope is in the equivalent time sampling mode, the AUTO command has no effect. However, if you change the sampling mode to real time or peak detect sampling the control will show the new value.

**Example**

This example changes the sampling rate to manual.

```
10  OUTPUT 707;":ACQUIRE:SRATE:AUTO OFF"
20  END
```

Query

```
:ACQuire:SRATe:AUTO?
```

The :ACQuire:SRATe:AUTO? query returns the current acquisition sample rate.

Returned Format

```
[:ACQuire:SRATe:AUTO] {1 | 0}<NL>
```

**Example**

This example places the current sample rate in the variable, Sample, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":ACQUIRE:SRATE:AUTO?"
30  ENTER 707;Sample
40  PRINT Sample
50  END
```

11

# Calibration Commands

# Calibration Commands

This chapter briefly explains the calibration of the Infiniium-Series Oscilloscopes. It is intended to give you and the calibration lab personnel an understanding of the calibration procedure and how the calibration subsystem is intended to be used. Also, this section acquaints you with the terms used in this chapter, and with help screens and data sheets.

A calibration procedure is included at the end of this chapter.

## Oscilloscope Calibration

Oscilloscope calibration establishes calibration factors for the oscilloscope. These factors are stored on the oscilloscope's hard disk.

- Initiate the calibration from the "Utilities Calibration" menu.

  You should calibrate the oscilloscope periodically (at least annually), or if the ambient temperature since the last calibration has changed more than ±5 °C. The temperature change since the last calibration is shown on the calibration status screen which is found under the "Utilities Calibration" dialog. It is the line labeled "Current Frame Temperature Δ: _ °C."

  To perform the oscilloscope calibration, you need a short BNC-to-BNC cable such as the 8120-1838 cable. When you initiate the calibration, instructions appear on the screen describing how to perform the calibration.

**See Also**

The Infiniium-Series Oscilloscopes Service Guide has more details about the mainframe calibration.

# Probe Calibration

Probe calibration establishes the gain and offset of a probe that is connected to a channel of the oscilloscope, and applies these factors to the calibration of that channel.

- **Initiate probe calibration from the "Utilities Calibration" menu.**

  To achieve the specified accuracy (±2%) with a probe connected to a channel, make sure the oscilloscope is calibrated.

  - For active probes that the oscilloscope can identify through the probe power connector, like the 54701A, the oscilloscope automatically adjusts the vertical scale factors for that channel even if a probe calibration is not performed.
  - For passive probes or nonidentified probes, the oscilloscope adjusts the vertical scale factors only if a probe calibration is performed.

- **If you do not perform a probe calibration but want to use a passive probe, enter the attenuation factor in the Probe Cal dialog under the Channel dialog.**

  - If the probe being calibrated has an attenuation factor that allows the oscilloscope to adjust the gain (in hardware) to produce even steps in the vertical scale factors, the oscilloscope will do so.
  - If the probe being calibrated has an unusual attenuation, like 3.75, the oscilloscope may have to adjust the vertical scale factors to an unusual number, like 3.75 V/div.

  Typically, probes have standard attenuation factors such as divide by 10, divide by 20, or divide by 100.

# Calibration Commands

The commands in the CALibration subsystem initiate the oscilloscope calibration over GPIB. These CALibration commands and queries are implemented in the Infiniium Oscilloscopes:

- CANCel
- CONTinue
- MPRotect
- OUTPut
- SDONe?
- SKEW
- STARt
- STATus?

.

**Let the Oscilloscope Warm Up First**

Let the oscilloscope warm up at least 30 minutes before you calibrate it.

# CANCel

Command

```
:CALibrate:CANCel
```

The :CALibrate:CANCel command cancels the calibration on the oscilloscope. If a calibration has been initiated by the :CALibrate:STARt command, this will cancel the calibration.

**Example**

This example cancels the oscilloscope calibration.

```
10  OUTPUT 707;":CALIBRATE:CANCEL"
20  END
```

## CONTinue

Command

```
:CALibrate:CONTinue
```

The :CALibrate:CONTinue command continues the calibration on the oscilloscope. If a calibration has been initiated by the :CALibrate:STARt command, this will continue through the next calibration step.

**Example**

This example continues the oscilloscope calibration.

```
10  OUTPUT 707;":CALIBRATE:CONTINUE"
20  END
```

# MPRotect

Command

```
:CALibrate:MPRotect {{ON|1} | {OFF|0}}
```

The :CALibrate:MPRotect command turns the calibration memory protection on or off. A calibration cannot be started with MPRotect on. This lets you protect the oscilloscope's calibration factors from accidentally being changed.

**Example**

This example turns on the calibration memory protection.

```
10  OUTPUT 707;":CALIBRATE:MPROTECT ON"
20  END
```

Query

```
:CALibrate:MPRotect?
```

The :CALibrate:MPRotect? query returns the current calibration memory protection status.

Returned Format

```
[:CALibrate:MPRotect] {ON | OFF}
```

**Example**

This example places the current selection for the calibration memory protection to be printed in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10  DIM Selection$[50]!Dimension variable
20  OUTPUT 707;":CALIBRATE:MPROTECT?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

# OUTPut

Command

```
:CALibrate:OUTPut {{AC|TRIGOUT} | {DC,<dc_value>}}
```

The :CALibrate:OUTPut command sets the coupling frequency, trigger output pulse, and dc level of the calibrator waveform output through the front panel CAL connector. To trigger other instruments, use the TRIGOUT setting to cause the oscilloscope to send a pulse when the trigger event occurs.

`<dc_value>` A real number for the DC level value in volts, adjustable from -2.5 V to +2.5 V DC.

**Example** This example puts a DC voltage of 2.0 V on the oscilloscope Aux Out connector.

```
10  OUTPUT 707;":CALIBRATE:OUTPUT DC,2.0"
20  END
```

Query

```
:CALibrate:OUTPut?
```

The :CALibrate:OUTPut? query returns the current setup.

Returned Format

```
[:CALibrate:OUTPut] {{AC|TRIGOUT} | {DC,<dc_value>}}
```

**Example** This example places the current selection for the DC calibration to be printed in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10  DIM Selection$[50]!Dimension variable
20  OUTPUT 707;":CALIBRATE:OUTPUT?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

## SDONe?

Query

```
:CALibrate:SDONe?
```

The :CALibrate:SDONe? (Step DONe) query will return when the current calibration step is complete.

The returned string contents tell you the next step. For example, "Connect Aux Out to Channel 1 with a short 50-ohm cable."

Returned Format

```
[:CALibrate:SDONe] <string>
```

**Example**

This example places the current selection for the calibration pass/fail status to be printed in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10  DIM Selection$[80]!Dimension variable
20  OUTPUT 707;":CALIBRATE:SDONE?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

# SKEW

**Command**

```
:CALibrate:SKEW {CHANnel<N> | EXTernal},<skew_value>
```

The :CALibrate:SKEW command sets the channel-to-channel skew factor for a channel. The numeric argument is a real number in seconds, which is added to the current time base position to shift the position of the channel's data in time. Use this command to compensate for differences in the electrical lengths of input paths due to cabling and probes.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<skew_value>` A real number, in seconds.

**Example**

This example sets the oscilloscope channel 1 skew to 0.1 S.

```
10  OUTPUT 707;":CALIBRATE:SKEW CHANNEL1,0.1S "
20  END
```

**Query**

```
:CALibrate:SKEW? {CHANnel<N>|EXTernal}
```

The :CALibrate:SKEW? query returns the current skew value.

**Returned Format**

```
[:CALibrate:SKEW] <skew_value><NL>
```

# STARt

Command

```
:CALibrate:STARt
```

The :CALibrate:STARt command starts the calibration sequence. Now :SDONe?, :CONTinue, and :CANCel are valid.

**Example**

This example starts the oscilloscope calibration.

```
10  OUTPUT 707;":CALIBRATE:START"
20  END
```

## STATus?

Query

```
:CALibrate:STATus?
```

The :CALibrate:STATus? query returns the calibration status of the oscilloscope. These are ten, comma-separated integers, with 1, 0, or -1. A "1" indicates pass, a "0" indicates fail and a "-1" indicates unused. This matches the status in the Calibration dialog box in the Utilities menu.

Returned Format

```
[:CALibrate:STATus] <status>
```

<status> <Frame Status>,
<Channel1 Vertical>, <Channel1 Trigger>,
<Channel2 Vertical>, <Channel2 Trigger>,
<Channel3 Vertical>, <Channel3 Trigger>, (-1 for 54810A, 54820A)
<Channel4 Vertical>, <Channel4 Trigger>, (-1 for 54810A, 54820A)
<Aux Trigger> (<Ext Trigger> for 54810A, 54820A)

12

# Channel Commands

# Channel Commands

The CHANnel subsystem commands control all vertical (Y axis) functions of the oscilloscope. You may toggle the channel displays on and off with the root level commands :VIEW and :BLANk, or with :CHANnel:DISPlay.

These CHANnel commands and queries are implemented in the Infiniium Oscilloscopes:

- BWLimit
- DISPlay
- INPut
- OFFSet
- PROBe
- PROBe:ATTenuation (only for the 1154A probe)
- PROBe:EADapter
- PROBe:ECoupling
- PROBe:EGAin
- PROBe:EOFFset
- PROBe:GAIN (only for the 1154A probe)
- PROBe:ID?
- PROBe:SKEW
- PROTection (only for the 54846A, 54845A, and 54835A)
- RANGe
- SCALe
- UNITs

54810A and 54820A Infiniium Oscilloscopes each have 2 channels. All other Infiniium Oscilloscope models have 4 channels.

# BWLimit

Command

```
:CHANnel<N>:BWLimit {{ON|1} | {OFF|0}}
```

The :CHANnel<N>:BWLimit command controls the low-pass filter, except on the 54846A, 54845A, and 54835A oscilloscopes, which has no low-pass filter. The 54846A, 54845A, and 54835A will not recognize this command or the query.

When ON, the bandwidth of the specified channel is limited. The bandwidth limit filter can be used with either AC or DC coupling.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**Example**

This example sets the internal low-pass filter to "ON" for channel 1.

```
10 OUTPUT 707;":CHANNEL1:BWLIMIT ON"
20 END
```

Query

```
:CHANnel<N>:BWLimit?
```

The :CHANnel<N>:BWLimit? query returns the state of the low-pass filter for the specified channel.

Returned Format

```
[:CHANnel<N>:BWLimit] {1|0}<NL>
```

**Example**

This example places the current setting of the low-pass filter in the variable Limit, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:BWLIMIT?"
30  ENTER 707;Limit
40  PRINT Limit
50  END
```

# DISPlay

Command

```
:CHANnel<N>:DISPlay {{ON|1} | {OFF|0}}
```

The :CHANnel<N>:DISPlay command turns the display of the specified channel on or off.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**Example**

This example sets channel 1 display to on.

```
10 OUTPUT 707;"CHANNEL1:DISPLAY ON"
20 END
```

Query

```
:CHANnel<N>:DISPlay?
```

The :CHANnel<N>:DISPlay? query returns the current display condition for the specified channel.

Returned Format

```
[:CHANnel<N>:DISPlay] {1|0}<NL>
```

**Example**

This example places the current setting of the channel 1 display in the variable Display, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:DISPLAY?"
30  ENTER 707;Display
40  PRINT Display
50  END
```

# INPut

Command

```
:CHANnel<N>:INPut <parameter>
```

The :CHANnel<N>:INPut command selects the input coupling, impedance, and LF/HF reject for the specified channel. The coupling for each channel can be AC, DC, DC50, or DCFifty when no probe is attached. If you have an 1153A probe attached, the valid parameters are DC, LFR1, and LFR2 (low-frequency reject).

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<parameter>` The parameters available in this command for Infiniium are.

- DC: DC coupling, 1 MΩ input impedance
- DC50 | DCFifty: DC coupling, 50Ω input impedance
- AC: AC 1 MΩ input impedance
- LFR1 | LFR2: AC 1 MΩ input impedance

**Example**

This example sets the channel 1 input to DC50.

```
10 OUTPUT 707;":CHANNEL1:INPut DC50"
20 END
```

Query

```
:CHANnel<N>:INPut?
```

The :CHANnel<N>:INPut? query returns the selected channel input parameter.

Returned Format

```
[CHANnel<N>:INPut]<parameter><NL>
```

**Example**

This example puts the current input for channel 1 in the string variable, Input$. The program then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"
20  OUTPUT 707;":CHANNEL1:INPUT?
30  ENTER 707;Input$
40  PRINT Input$
50  END
```

# OFFSet

Command

```
:CHANnel<N>:OFFSet <offset_value>
```

The :CHANnel<N>:OFFSet command sets the voltage that is represented at the center of the display for the selected channel. Offset parameters are probe and vertical scale dependent.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

<offset _value> A real number for the offset value at center screen. Usually expressed in volts, but can be in other measurement units, such as amperes, if you have specified other units using the :CHANnel<N>:UNITs command.

**Example**

This example sets the offset for channel 1 to 0.125 in the current measurement units:

```
10  OUTPUT 707;":CHANNEL1:OFFSET 125E-3"
20  END
```

Query

```
:CHANnel<N>:OFFSet?
```

The :CHANnel<N>:OFFSet? query returns the current offset value for the specified channel.

Returned Format

```
[CHANnel<N>:OFFSet] <offset_value><NL>
```

**Example**

This example places the offset value of the specified channel in the string variable, Offset$, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;"SYSTEM:HEADER OFF"
20 OUTPUT 707;"CHANNEL1:OFFSET?"
30 ENTER 707;Offset
40 PRINT Offset
50 END
```

# PROBe

**Command**

```
:CHANnel<N>:PROBe <attenuation_factor>[,{RATio | DECibel}]
```

The :CHANnel<N>:PROBe command sets the probe attenuation factor and, optionally, the units for the probe attenuation factor. The range of the probe attenuation factor is from 0.0001 to 1,000,000 and from –80 dB to 120 dB. The reference factors that are used for scaling the display are changed with this command, and affect automatic measurements and trigger levels.

The ",DEC" or ",RAT" also sets the "mode" for the probe attenuation. This mode also determines the units that may be used for a subsequent command. For example, if you select RATio mode, "DB" cannot be used. In "DECibel" mode, you can specify the units for the argument as "DB".

`<N>` An integer, 1-2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1-4, for all other Infiniium Oscilloscope models.

`<attenuation_factor>` A real number from 0.0001 to 1,000,000 or from –80 dB to 120 dB, representing the probe attenuation factor. The factor depends on the units.

**Example**

This example sets the probe attenuation factor of channel 1 to 10, and the units to decibel.

```
10  OUTPUT 707;":CHANNEL1:PROBE 10,DEC"
20  END
```

**See Also**

For information on skew, see the Calibration Commands.

Query

```
:CHANnel<N>:PROBe?
```

The :CHANnel<N>:PROBe? query returns the current probe attenuation setting for the selected channel and the units.

Returned Format

```
[:CHANnel<N>:PROBe] <attenuation>,{RATio | DECibel}<NL>
```

**Example**

This example places the current attenuation setting for channel 1 in the string variable, Atten$, then the program prints the contents.

```
10  DIM Atten$[50]!Dimension variable
20  OUTPUT 707;":CHANNEL1:PROBE?
30  ENTER 707;Atten$
40  PRINT Atten$
50  END
```

If you use a string variable, the query returns the attenuation value and the factor (decibel or ratio). If you use an integer variable, the query returns the attenuation value. You must then read the attenuation units into a string variable.

## PROBe:ATTenuation

Command

```
:CHANnel<N>:PROBe:ATTenuation {DIV1 | DIV10}
```

The :CHANnel<N>:PROBe:ATTenuation command sets the probe's attenuation. There are some Infiniium active and differential probes that have the ability to change the probe's input amplifier's attenuation.

This command is only available when an Infiniium active or differential probe is connected to the channel. If one of these probes is not connected to the channel you will get a settings conflict error.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**Example**

This example sets the probe attenuation for channel 1 to divide by 10.

```
10  OUTPUT 707;":CHANNEL1:PROBE:ATTENUATION DIV10"
20  END
```

Query

```
:CHANnel<N>:PROBe:ATTenuation?
```

The :CHANnel<N>:PROBe:ATTenuation? query returns the current probe attenuation setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:ATTenuation] {DIV1 | DIV10}<NL>
```

# PROBe:EADapter

Command

```
:CHANnel<N>:PROBe:EADapter {NONE | DIV10 |
DIV20 | DIV100}
```

The :CHANnel<N>:PROBe:EADapter command sets the Infiniium external adapter control. There are some probes that have external adapters that you can attach to the end of your probe. When you attach one of these adapters, you should use the EADapter command to set the external adapter control to match the adapter connected to your probe as follows.

| Parameter | Description |
|---|---|
| NONE | Use this setting when there is no adapter connected to the end of your probe. |
| DIV10 | Use this setting when you have a divide by 10 adapter connected to the end of your probe. |
| DIV20 | Use this setting when you have a divide by 20 adapter connected to the end of your probe. |
| DIV100 | Use this setting when you have a divide by 100 adapter connected to the end of your probe. |

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**Example**

This example sets the external adapter for channel 1 to divide by 10:

```
10  OUTPUT 707;":CHANNEL1:PROBE:EADAPTER DIV10"
20  END
```

Query

```
:CHANnel<N>:PROBe:EADapter?
```

The :CHANnel<N>:PROBe:EADapter? query returns the current external adapter value for the specified channel.

Returned Format

```
[CHANnel<N>:PROBe:EDApter] {NONE | AC | DIV10 | DIV20
| DIV100}<NL>
```

**Example**

This example places the external adapter value of the specified channel in the string variable, Adapter$, then prints the contents of the variable to the computer's screen.

```
10  DIM Adapter$[50]!Dimension variable
20  OUTPUT 707;":CHANNEL1:PROBE:EADAPTER?
30  ENTER 707;Adapter$
40  PRINT Adapter$
50  END
```

# PROBe:ECoupling

Command

```
:CHANnel<N>:PROBe:ECoupling {NONE | AC}
```

The :CHANnel<N>:PROBe:ECoupling command sets the Infiniium external coupling adapter control. There are some probes that have external coupling adapters that you can attach to the end of your probe. When you attach one of these adapters, you should use the ECoupling command to set the external coupling adapter control to match the adapter connected to your probe as follows.

| Parameter | Description |
|---|---|
| NONE | Use this setting when there is no adapter connected to the end of your probe. |
| AC | Use this setting when you have an ac coupling adapter connected to the end of your probe. |

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**Example**

This example sets the external coupling adapter for channel 1 to ac:

```
10  OUTPUT 707;":CHANNEL1:PROBE:ECOUPLING AC"
20  END
```

Query

```
:CHANnel<N>:PROBe:ECoupling?
```

The :CHANnel<N>:PROBe:ECoupling? query returns the current external coupling adapter value for the specified channel.

Returned Format

```
[CHANnel<N>:PROBe:ECoupling] {NONE | AC}<NL>
```

**Example**

This example places the external coupling adapter value of the specified channel in the string variable, Adapter$, then prints the contents of the variable to the computer's screen.

```
10  DIM Adapter$[50]!Dimension variable
20  OUTPUT 707;":CHANNEL1:PROBE:ECOUPLING?
30  ENTER 707;Adapter$
40  PRINT Adapter$
50  END
```

# PROBe:EGAin

Command

```
:CHANnel<N>:PROBe:EGAin <gain_value>
```

The :CHANnel<N>:PROBe:EGAin command sets the probe gain. The units of volts, amperes, watts, and unknown are set using the :CHANnel<N>:UNITs command.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<gain_value>` A real number for the gain value.

**Example**

This example sets the probe gain for channel 1 to 125 x $10^{-3}$.

```
10  OUTPUT 707;":CHANNEL1:PROBE:EGAIN 125E-3"
20  END
```

Query

```
:CHANnel<N>:PROBe:EGAin?
```

The :CHANnel<N>:PROBe:EGAin? query returns the gain setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:EGAin] <gain_value><NL>
```

# PROBe:EOFFset

Command

```
:CHANnel<N>:PROBe:EOFFset <offset_value>
```

The :CHANnel<N>:PROBe:EOFFset command sets the probe offset. The units of volts, amperes, watts, and unknown are set using the :CHANnel<N>:UNITs command.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<offset_value>` A real number for the offset value.

**Example**

This example sets the probe offset for channel 1 to 125 x $10^{-3}$.

```
10  OUTPUT 707;":CHANNEL1:PROBE:EOFFSET 125E-3"
20  END
```

Query

```
:CHANnel<N>:PROBe:EOFFset?
```

The :CHANnel<N>:PROBe:EOFFset? query returns the offset value for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:EOFFset] <offset_value><NL>
```

# PROBe:GAIN

Command

```
:CHANnel<N>:PROBe:GAIN {X1 | X10}
```

The :CHANnel<N>:PROBe:GAIN command sets the probe gain. There are some Infiniium active and differential probes that have the ability to change the probe's input amplifier gain.

This command is only available when an Infiniium active or differential probe is connected to the channel. If one of these probes is not connected to the channel you will get a settings conflict error.

The units of volts, amperes, watts, and unknown are set using the :CHANnel<N>:UNITs command.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**Example**

This example sets the probe gain for channel 1 to times 10.

```
10  OUTPUT 707;":CHANNEL1:PROBE:GAIN X10"
20  END
```

Query

```
:CHANnel<N>:PROBe:GAIN?
```

The :CHANnel<N>:PROBe:GAIN? query returns the current probe gain setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:GAIN] {X1 | X10}<NL>
```

# PROBe:ID?

**Query** `:CHANnel<N>:PROBe:ID?`

The :CHANnel<N>:PROBe:ID? query returns the type of probe attached to the specified oscilloscope channel.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**Returned Format** [:CHANnel<N>:PROBe:ID] <probe_id>

`<probe_id>` A string of up to 9 alphanumeric characters. Some of the possible returned values are:

- 1147A
- 1154A
- 1159A
- AutoProbe
- E2621A
- E2622A
- HP1152A
- HP1153A
- NONE
- Probe

**Example** This example reports the probe type connected to channel 1, if one is connected.

```
10  OUTPUT 707;":CHANNEL1:PROBE:ID?"
20  END
```

# PROBe:SKEW

Command

```
:CHANnel<N>:PROBe:SKEW <skew_value>
```

The :CHANnel<N>:PROBe:SKEW command sets the channel-to-channel skew factor for the specified channel. You can use the oscilloscope's probe skew control to remove timing differences between probes or cables on different channels.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<skew_value>` A real number for the skew value, in the range -100 µs to 100 µs.

**Example**

This example sets the probe skew for channel 1 to 10 µs.

```
10  OUTPUT 707;":CHANNEL1:PROBE:SKEW 10E-6"
20  END
```

Query

```
:CHANnel<N>:PROBe:SKEW?
```

The :CHANnel<N>:PROBe:SKEW? query returns the current probe skew setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:SKEW] <skew_value><NL>
```

# PROTection:CLEar

Command

```
:CHANnel<N>:PROTection:CLEar
```

The :CHANnel<N>:PROTection:CLEar command is used to clear (reset) the overload protection. It allows the channel to be used again after the waveform that caused the overload has been removed from the channel input. This command is only available on the 54846A, 54845A, and 54835A.

<N> An integer, 1 - 4.

**Example**

This example clears the overload protection for channel 1.

```
10  OUTPUT 707;":CHANNEL1:PROTECTION:CLEAR"
20  END
```

# PROTection?

Command

```
:CHANnel<N>:PROTection?
```

The :CHANnel<N>:PROTection? query returns the state of the input protection for CHANnel<N>. If the channel protection is engaged, then a 1 is returned otherwise a 0 is returned. This command is only available on the 54846A, 54845A, and 54835A.

<N> An integer, 1 - 4.

Returned Format

```
[:CHANnel<N>:PROTection] {1 | 0}
```

**Example**

This example places the current state of the input protection for the specified channel in the number variable, Protect, then prints the contents of the variable to the computer's screen.

```
10  OUPUT 707;"SYSTEM:HEADER OFF" !Response headers off
20  OUTPUT 707;":CHANNEL1:PROTECTION?"
30  ENTER 707;Protect
40  PRINT Protect
50  END
```

# RANGe

**Command**

```
:CHANnel<N>:RANGe <range_value>
```

The :CHANnel<N>:RANGe command defines the full-scale vertical axis of the selected channel. It sets up acquisition and display hardware to display the waveform at a given range scale. The values represent the full-scale deflection factor of the vertical axis in volts. These values change as the probe attenuation factor is changed.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<range_value>` A real number for the full-scale voltage of the specified channel number.

**Example**

This example sets the full-scale range for channel 1 to 500 mV.

```
10  OUTPUT 707;":CHANNEL1:RANGE 500E-3"
20  END
```

**Query**

```
:CHANnel<N>:RANGe?
```

The :CHANnel<N>:RANGe? query returns the current full-scale vertical axis setting for the selected channel.

**Returned Format**

```
[:CHANnel<N>:RANGe]<range_value><NL>
```

**Example**

This example places the current range value in the number variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":CHANNEL1:RANGE?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

# SCALe

Command

```
:CHANnel<N>:SCALe <scale_value>
```

The :CHANnel<N>:SCALe command sets the vertical scale, or units per division, of the selected channel. This command is the same as the front-panel channel scale.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<scale_value>` A real number for the vertical scale of the channel in units per division.

**Example**

This example sets the scale value for channel 1 to 500 mV.

```
10  OUTPUT 707;":CHANNEL1:SCALE 500E-3"
20  END
```

Query

```
:CHANnel<N>:SCALe?
```

The :CHANnel<N>:SCALe? query returns the current scale setting for the specified channel.

Returned Format

```
[:CHANnel<N>:SCALe] <scale_value><NL>
```

**Example**

This example places the current scale value in the number variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":CHANNEL1:SCALE?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

# UNITs

Command

```
:CHANnel<N>:UNITs {VOLT | AMPere | WATT | UNKNown}
```

The :CHANnel<N>:UNITs command sets the vertical units. You can specify Y-axis units of VOLTs, AMPs, WATTs, or UNKNown. The units are implied for other pertinent channel commands (such as :CHANnel<N>:RANGe and :CHANnel<N>:OFFSet). See the Probe Setup dialog box for more information.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**Example**

This example sets the units for channel 1 to amperes.

```
10  OUTPUT 707;":CHANNEL1:UNITS AMPERE"
20  END
```

Query

```
:CHANnel<N>:UNITs?
```

The :CHANnel<N>:UNITs? query returns the current units setting for the specified channel.

Returned Format

```
[:CHANnel<N>:UNITs] {VOLT | AMPere | WATT | UNKNown}<NL>
```

**Example**

This example places the vertical units for the specified channel in the string variable, Units$, then prints the contents of the variable to the computer's screen.

```
10 DIM Units$[50]
20 OUTPUT 707;"CHANNEL1:UNITS?"
30 ENTER 707;Units$
40 PRINT Units$
50 END
```

13

# Disk Commands

# Disk Commands

The DISK subsystem commands perform the disk operations as defined in the File menu. This allows saving and loading of waveforms and setups, as well as saving screen images to bitmap files.

**Enclose File Name in Quotation Marks**

When specifying a file name, you must enclose it in quotation marks.

**Filenames are Not Case Sensitive.**

The filename that you use is not case sensitive.

These DISK commands and queries are implemented in the Infiniium Oscilloscopes:

- CDIRectory
- DELete
- DIRectory?
- LOAD
- MDIRectory
- PWD?
- SIMage
- STORe

# CDIRectory

Command

```
:DISK:CDIRectory "<directory>"
```

The :DISK:CDIRectory command changes the present working directory to the designated directory name. An error occurs when the requested directory does not exist. You can then view the error with the :SYSTem:ERRor? [{NUMBer | STRing}] query.

`<directory>` A character-quoted ASCII string, which can include the subdirectory designation. You must separate the directory name and any subdirectories with a backslash (\).

**Example**

This example sets the present working directory to C:\SCOPE\DATA.

```
10 OUTPUT 707;":DISK:CDIRECTORY ""C:\SCOPE\DATA"""
20 END
```

**Directories Not Allowed**

You can execute the command CDIR "A:\", but the following commands are not allowed.

:DISK:CDIR "C:\"

:DISK:CDIR "C:\SCOPE\BIN"

:DISK:CDIR "C:\SCOPE\CAL"

If you attempt to execute CDIR using these directories an eror message (-257) is issued and the present working directory (PWD) is unchanged.

# DELete

Command

```
:DISK:DELete "<file_name>"
```

The :DISK:DELete command deletes a file from the disk. An error is displayed on the oscilloscope screen if the requested file does not exist.

`<file_name>` A character-quoted ASCII string which can include subdirectories with the name of the file.

**Example**

This example deletes FILE1.SET from the disk.

```
10 OUTPUT 707;":DISK:DELETE ""FILE1.SET"""
20 END
```

# DIRectory?

**Query** `:DISK:DIRectory? ["<directory>"]`

The :DISK:DIRectory? query returns the requested directory listing. Each entry is 63 bytes long, including a carriage return and line feed.

`<directory>` The list of filenames and directories.

**Returned Format** `[:DISK:DIRectory]<n><NL><directory>`

`<n>` The specifier that is returned before the directory listing, indicating the number of lines in the listing.

`<directory>` The list of filenames and directories. Each line is separated by a <NL>.

**Example**

This example displays a number, then displays a list of files and directories in the current directory. The number indicates the number of lines in the listing.

```
10  DIM A$[80]
20  INTEGER Num_of_lines
30  OUTPUT 707;":DISK:DIR?"
40  ENTER 707;Num_of_lines
50  PRINT Num_of_lines
60  FOR I=1 TO Num_of_lines
70  ENTER 707;A$
80  PRINT A$
90  NEXT I
100 END
```

# LOAD

Command

```
:DISK:LOAD "<file_name>"[,<destination>]
```

The :DISK:LOAD command restores a setup or a waveform from the disk. The type of file is determined by the filename suffix if one is present, or by the destination field if one is not present. You can load .wfm, .txt, and .set file types. The destination is only used when loading a waveform memory.

`<file_name>` A quoted ASCII string with a maximum of 254 characters including the entire path name, if used. You can use either .wfm, .txt, or .set as a suffix after the filename. If no file suffix is specified, the default is .wfm.

The present working directory is assumed, or you can specify the entire path. For example, you can load the standard setup file "setup0.set" using the command:

:DISK:LOAD "c:\scope\setups\setup0.set"

Or, you can use :DISK:CDIRectory to change the present working directory to c:\scope\setups, then just use the file name ("setup0.set", for example).

`<destination>` WMEMory<N>.

Where <N> is an integer from 1-4.

If a destination is not specified, waveform memory 1 is used.

**Example**

This example restores the waveform in FILE1.WFM to waveform memory 1.

```
10 OUTPUT 707;":DISK:LOAD ""FILE1.WFM"",WMEM1"
20 END
```

# MDIRectory

**Command**

```
:DISK:MDIRectory "<directory>"
```

The :DISK:MDIRectory command creates a directory in the present working directory, with the designated directory name. An error is displayed if the requested subdirectory does not exist.

`<directory>` A quoted ASCII string which can include subdirectories. You must separate the directory name and any subdirectories with a backslash (\).

**Example**

This example creates the directory CPROGRAMS in the present working directory.

```
10 OUTPUT 707;":DISK:MDIRECTORY ""C:\SCOPE\DATA\CPROGRAMS"""
20 END
```

You can check your path with the :DISK:DIRectory? query.

# PWD?

Query

```
:DISK:PWD?
```

The :DISK:PWD? query returns the name of the present working directory (including the full path).

Returned Format

```
:DISK:PWD? <present_working_directory><NL>
```

**Example**

This example places the present working directory in the string variable Wdir?, then prints the contents of the variable to the computer's screen.

```
10 DIM Wdir$[200]
20 OUTPUT 707;":DISK:PWD?"
30 ENTER 707; Wdir$
40 PRINT Wdir$
50 END
```

## SIMage

**Command**

```
:DISK:SIMage "<file_name>" [,<format>
[,{SCReen|GRATicule}
[,{ON|1} | {OFF|0}
[,{NORMal|INVert}]]]]
```

The DISK:SIMage command saves a screen image in BMP, PCX, PS, EPS, GIF or TIF format. The extension is supplied by the oscilloscope depending on the selected file format. If you do not include the format in the command, the file is saved in the format which is shown in the Save Screen dialog box.

`<file_name>` A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

`<format>` {BMP | PCX | PS | EPS | GIF | TIF}

**Examples**

```
OUTPUT 707;":DISK:SIM " "FILE1" ", PCX, ON, INVERT"
or
OUTPUT 707;":DISK:SIM " "FILE1" ", TIF, ON"
or
OUTPUT 707;":DISK:SIM " "FILE1" " "
```

# STORe

Command

```
:DISK:STORe <source>,"<file_name>"
[,<format>[,<preamble>]]
```

The :DISK:STORe command saves a setup or a waveform to a disk. The filename does not include a suffix. The suffix is supplied by the oscilloscope, depending on the source and file format specified.

`<source>` {CHANnel<N> | FUNCtion<N> | HISTogram | WMEMory<N> | SETup}

`<N>` For CHANnel<N>:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

For FUNCtion<N> and WMEM<N>:

An integer, 1 - 4, representing the function or waveform memory number.

`<file_name>` A quoted ASCII string with a maximum of 254 characters including the entire path name, if used. The filename assumes the present working directory if a path does not precede the file name.

`<format>` One of {INTernal | TEXT {,YVALues | VERBose | XYPairs | TSV}}

`<preamble>` {ON | OFF}

**Fields and Default Values**

The format field is for waveforms, and the default is INTernal. In TEXT mode, you may specify Y values so that only the Y values are stored. VERBose is the default in which Y values and the waveform preamble are saved. Only waveforms of 128K or less can be written to disk in the TEXT formats. See the Waveform Commands chapter for information on converting data to values.

**Example**

This example stores the current oscilloscope setup to FILE1 on the disk.

```
10 OUTPUT 707;":DISK:STORE SETUP,""FILE1"""
20 END
```

14

# Display Commands

# Display Commands

The DISPlay subsystem controls the display of data, text, and graticules, and the use of color.

These DISPlay commands and queries are implemented in the Infiniium Oscilloscopes:

- CGRade
- CGRade:LEVels?
- COLumn
- CONNect
- DATA?
- DCOLor (Default COLor)
- GRATicule
- LINE
- PERSistence
- ROW
- SCOLor (Set COLor)
- SSAVer
- SSAVer:AFTer
- STRing
- TEXT

# CGRade

**Command**

```
:DISPlay:CGRade {{ON | 1} | {OFF | 0}}
```

The :DISPlay:CGRade command sets the color grade persistence on or off.

When in the color grade persistence mode, all waveforms are mapped into a database and shown with different colors representing varying number of hits in a pixel. "Connected dots" display mode is disabled when the color grade persistence is on.

The oscilloscope has three features that use a specific database. This database uses a different memory area than the waveform record for each channel. The three features that use the database are histograms, mask testing, and color grade persistence. When any one of these three features is turned on, the oscilloscope starts building the database. The database is the size of the graticule area and varies in size. Behind each pixel is a 21-bit counter. Each counter is incremented each time a pixel is hit by data from a channel or function. The maximum count (saturation) for each counter is 2,097,151. You can check to see if any of the counters is close to saturation by using the DISPlay:CGRade:LEVels? query. The color grade persistence uses colors to represent the number of hits on various areas of the display. The default color-grade state is off.

**Example**

This example sets the color grade persistence on.

```
10 OUTPUT 707;":DISPLAY:CGRADE ON"
20 END
```

Query

```
:DISPlay:CGRade?
```

The DISPlay:CGRade query returns the current color-grade state.

Returned Format

```
[:DISPlay:CGRade] {ON | OFF}<NL>
```

**Example**

This example returns the current color grade state.

```
10 DIM Setting$[50]          !Dimension variable
20 OUTPUT 707;":DISPLAY:CGRADE?"
30 ENTER 707;Cgrade$
40 PRINT Cgrade$
50 END
```

## CGRade:LEVels?

**Query** `:DISPlay:CGRade:LEVels?`

The :DISPlay:CGRade:LEVels? query returns the range of hits represented by each color. Fourteen values are returned, representing the minimum and maximum count for each of seven colors. The values are returned in the following order:

- White minimum value
- White maximum value
- Yellow minimum value
- Yellow maximum value
- Orange minimum value
- Orange maximum value
- Red minimum value
- Red maximum value
- Pink minimum value
- Pink maximum value
- Blue minimum value
- Blue maximum value
- Green minimum value
- Green maximum value

**Returned Format** `[DISPlay:CGRade:LEVels] <color format><NL>`

`<color format>` <intensity color min/max> is an integer value from 0 to 2,076,151

**Example**

This example gets the range of hits represented by each color and prints it on the computer screen:

```
10 DIM Setting$[50]          !Dimension variable
20 OUTPUT 707;":DISPLAY:CGRADE:LEVELS?"
30 ENTER 707;Cgrade$
40 PRINT Cgrade$
50 END
```

Colors start at green minimum, maximum, then blue, pink, red, orange, yellow, white. The format is a string where commas separate minimum and maximum values. The largest number in the string can be 2,076,151

An example of a possible returned string is as follows:

1,414,415,829,830,1658,1659,3316,3317,6633,6634,13267,13268,26535

# COLumn

Command

```
:DISPlay:COLumn <column_number>
```

The :DISPlay:COLumn command specifies the starting column for subsequent :DISPlay:STRing and :DISPlay:LINE commands.

`<column_number>` An integer, 0 to 81, representing the starting column for subsequent :DISPlay:STRing and :DISPlay:LINE commands. The entire viewing area of the screen is divided into a maximum of 31 lines, depending on the size of the waveform area.

**Example**

This example sets the starting column for subsequent :DISPlay:STRing and :DISPlay:LINE commands to column 10.

```
10  OUTPUT 707;":DISPLAY:COLUMN 10"
20  END
```

Query

```
:DISPlay:COLumn?
```

The :DISPlay:COLumn? query returns the column where the next :DISPlay:LINE or :DISPlay:STRing starts.

Returned Format

```
[:DISPlay:COLumn] <value><NL>
```

**Example**

This example returns the current column setting to the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":DISPLAY:COLUMN?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

## CONNect

Command

```
:DISPlay:CONNect {{ON|1} | {OFF|0}}
```

When enabled, :DISPlay:CONNect draws a line between consecutive waveform data points. This is also known as linear interpolation.

**Example**

This example turns on the connect-the-dots feature.

```
10  OUTPUT 707;":DISPLAY:CONNECT ON"
20  END
```

Query

```
:DISPlay:CONNect?
```

The :DISPlay:CONNect? query returns the status of the connect-the-dots feature.

Returned Format

```
[:DISPlay:CONNect] {{ON|1} | {OFF|0}}<NL>
```

# DATA?

**Query**

```
:DISPlay:DATA?
[<type>[,<screen_mode>[,<compression>
[,<inversion>]]]]
```

`<type>` The file type: BMP | PCX | EPS | PS | GIF | TIF.

`<screen_mode>` The display setting: SCReen | GRATicule. Selecting GRATicule displays a 10-by-8 (unit) display graticule on the screen. See also :DISPlay:GRATicule.

`<compression>` The file compression feature: ON | OFF.

`<inversion>` The inversion of the displayed file: NORMal | INVert.

The :DISPlay:DATA? query returns information about the captured data. If no options to the query are specified, the default selections are PCX file type, SCReen mode, compression turned ON, and inversion set to NORMal.

**Returned Format**

```
[:DISPlay:DATA] <binary_block_data><NL>
```

`<binary_block_data>` Data in the IEEE 488.2 definite block format.

# DCOLor

**Command**

```
:DISPlay:DCOLor [<color_name>]
```

The :DISPlay:DCOLor command resets the screen colors to the predefined factory default colors. It also resets the grid intensity.

`<color_name>` `{CGLevel1 | CGLevel2 | CGLevel3 | CGLevel4 | CGLevel5 | CGLevel6 | CGLevel7 | CHANnel1 | CHANnel2 | CHANnel3 | CHANnel4 | DBACkgrnd | GRID | MARKers | MEASurements | MIConsCGLevel1| MTPolygons | STEXt | WBACkgrnd | TINPuts | WOVerlap | TSCale | DHIGhlight | WMEMories | WINText | WINBackgrnd}`

**Example**

This example sends the :DISPlay:DCOLor command.

```
10  OUTPUT 707;":DISPLAY:DCOLOR"
20  END
```

# GRATicule

Commands

```
:DISPlay:GRATicule {GRID|FRAMe}
:DISPlay:GRATicule:INTensity <intensity_value>
:DISPlay:GRATicule:SPLit {{ON|1} {OFF|0}}
```

The :DISPlay:GRATicule command selects the type of graticule that is displayed. Infiniium oscilloscopes have a 10-by-8 (unit) display graticule grid GRID), a grid line is place on each vertical and horizontal division. When it is off (FRAMe), a frame with tic marks surrounds the graticule edges.

You can dim the grid's intensity or turn the grid off to better view waveforms that might be obscured by the graticule lines using the :DISPlay:GRATicule:INTensity command. Otherwise, you can use the grid to estimate waveform measurements such as amplitude and period.

When printing, the grid intensity control does not affect the hard copy. To remove the grid from a printed hard copy, you must turn off the grid before printing.

`<intensity_value>` A integer from 0 to 100, indicating the percentage of grid intensity.

You can divide the waveform viewing area into two separate viewing areas using the :DISPlay:GRATicule:SPLit command. This allows you to separate waveforms without having to adjust the vertical position controls.

**Example**

This example sets up the oscilloscope's display background with a frame that is separated into major and minor divisions.

```
10  OUTPUT 707;":DISPLAY:GRATICULE FRAME"
20  END
```

Queries

```
:DISPlay:GRATicule?
:DISPlay:GRATicule:INTensity?
```

The :DISPlay:GRATicule? and :DISPlay:GRATicule:INTensity? queries return the type of graticule currently displayed, or the intensity, depending on the query you request.

Returned Format

```
[:DISPlay:GRATicule] {GRID|FRAMe}<NL>
[:DISPlay:GRATicule:INTensity] <value><NL>
```

**Example**

This example places the current display graticule setting in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":DISPLAY:GRATICULE?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# LINE

Command

```
:DISPlay:LINE "<string_argument>"
```

The :DISPlay:LINE command writes a quoted string to the screen, starting at the location specified by the :DISPlay:ROW and :DISPlay:COLumn commands. When using the C programming language, quotation marks as shown in the example delimit a string.

`<string_argument>` Any series of ASCII characters enclosed in quotation marks.

**Example**

This example writes the message "Infiniium Test" to the screen, starting at the current row and column location.

```
10  OUTPUT 707;":DISPLAY:LINE ""Infiniium Test"""
20  END
```

This example writes the message "Infiniium Test" to the screen using C. Quotation marks are included because the string is delimited.

```
printf("\"Infiniium Test\"");
```

You may write text up to column 81. If the characters in the string do not fill the line, the rest of the line is blanked. If the string is longer than the space available on the current line, the excess characters are discarded.

In any case, the ROW is incremented and the COLumn remains the same. The next :DISPlay:LINE command will write on the next line of the display. After writing the last line in the display area, the ROW is reset to 0.

## PERSistence

Command

```
:DISPlay:PERSistence {MINimum | INFinite |
<persistence_value>}
```

The :DISPlay:PERSistence command sets the display persistence. It works in both real time and equivalent time modes. The parameter for this command can be either MINimum (zero persistence), INFinite, or a real number from 0.1 to 40.0, representing the persistence in seconds.

`<persistence_value>` A real number, 0.1 to 40.0, representing the persistence in seconds.

**Example**

This example sets the persistence to infinite.

```
10  OUTPUT 707;":DISPLAY:PERSISTENCE INFINITE"
20  END
```

Query

```
:DISPlay:PERSistence?
```

The :DISPlay:PERSistence? query returns the current persistence value.

Returned Format

```
[:DISPlay:PERSistence] {MINimum | INFinite | <value>}<NL>
```

**Example**

This example places the current persistence setting in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":DISPLAY:PERSISTENCE?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# ROW

Command

```
:DISPlay:ROW <row_number>
```

The :DISPlay:ROW command specifies the starting row on the screen for subsequent :DISPlay:STRing and :DISPlay:LINE commands. The row number remains constant until another :DISPlay:ROW command is received, or the row is incremented by the :DISPlay:LINE command.

`<row_number>` An integer, 0 to 31, representing the starting row for subsequent :DISPlay:STRing and :DISPlay:LINE commands. The entire screen viewing area is divided into a maximum of 31 lines, depending on the size of the waveform area.

**Example** This example sets the starting row for subsequent :DISPlay:STRing and :DISPlay:LINE commands to 10.

```
10  OUTPUT 707;":DISPLAY:ROW 10"
20  END
```

Query

```
:DISPlay:ROW?
```

The :DISPlay:ROW? query returns the current value of the row.

Returned Format

```
[:DISPlay:ROW] <row_number><NL>
```

**Example** This example places the current value for row in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":DISPLAY:ROW?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# SCOLor

Command

```
:DISPlay:SCOLor <color_name>, <hue>, <saturation>,
<luminosity>
```

The :DISPlay:SCOLor command sets the color of the specified display element and restores the colors to their factory settings. The display elements are described in Table 14-1.

<color_name> {CGLevel1 | CGLevel2 | CGLevel3 | CGLevel4 | CGLevel5 | CGLevel6 | CGLevel7 | CHANnel1 | CHANnel2 | CHANnel3 | CHANnel4 | DBACkgrnd | GRID | MARKers | MEASurements | MICons | MTPolygons | STEXt | WBACkgrnd | TINPuts | WOVerlap | TSCale | DHIGhlight | WMEMories | WINText | WINBackgrnd}

**Table 14-1 Color Names**

| Color Name | Definition |
|---|---|
| CGLevel1 | Color Grade Level 1 waveform display element. |
| CGLevel2 | Color Grade Level 2 waveform display element. |
| CGLevel3 | Color Grade Level 3 waveform display element. |
| CGLevel4 | Color Grade Level 4 waveform display element. |
| CGLevel5 | Color Grade Level 5 waveform display element. |
| CGLevel6 | Color Grade Level 6 waveform display element. |
| CGLevel7 | Color Grade Level 7 waveform display element. |
| CHANnel1 | Channel 1 waveform display element. |
| CHANnel2 | Channel 2 waveform display element. |
| CHANnel3 | Channel 3 waveform display element. |
| CHANnel4 | Channel 4 waveform display element. |
| DBACkgrnd | Display element for the border around the outside of the waveform viewing area. |
| GRID | Display element for the grid inside the waveform viewing area. |
| MARKers | Display element for the markers. |
| MEASurements | Display element for the measurements text. |
| MICons | Display element for measurement icons to the left of the waveform viewing area. |

| Color Name | Definition |
|---|---|
| MTPolygons | Display element for the mask test violation regions |
| STEXt | Display element for status messages displayed in the upper left corner of the display underneath the menu bar. Changing this changes the memory bar's color. |
| WBACkgrnd | Display element for the waveform viewing area's background. |
| TINPuts | Display element for line and aux menu entries on 54815/25/45A oscilloscopes. On 54810/20A oscilloscopes, it is the display element for line and external menu entries. |
| WOVerlap | Display element for waveforms when they overlap each other. |
| TSCale | Display element for horizontal scale and offset control text. |
| DHIGhlight | Display element for the highlighted waveform when in delayed sweep mode. |
| WMEMories | Display element for waveform memories. |
| WINText | Display element used in dialog box controls and pull-down menus. |
| WINBackgrnd | Display element for the background color used in dialog boxes and buttons. |

`<hue>` An integer from 0 to 100. The hue control sets the color of the chosen display element. As hue is increased from 0%, the color changes from red, to yellow, to green, to blue, to purple, then back to red again at 100% hue. For color examples, see the sample color settings table in the Infiniium Oscilloscope online help file. Pure red is 100%, pure blue is 67%, and pure green is 33%.

`<saturation>` An integer from 0 to 100. The saturation control sets the color purity of the chosen display element. The saturation of a color is the purity of a color, or the absence of white. A 100% saturated color has no white component. A 0% saturated color is pure white.

`<luminosity>` An integer from 0 to 100. The luminosity control sets the color brightness of the chosen display element. A 100% luminosity is the maximum color brightness. A 0% luminosity is pure black.

**Example**

This example sets the hue to 50, the saturation to 70, and the luminosity to 90 for the markers.

```
10  OUTPUT 707;":DISPLAY:SCOLOR MARKERS,50,70,90"
20  END
```

Query

```
:DISPlay:SCOLor? <color_name>
```

The :DISPlay:SCOLor? query returns the hue, saturation, and luminosity for the specified color.

Returned Format

```
[:DISPlay:SCOLor] <color_name>, <hue>, <saturation>,
<luminosity><NL>
```

**Example**

This example places the current settings for the graticule color in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":DISPLAY:SCOLOR? GRATICULE"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# SSAVer

**Commands**

```
:DISPlay:SSAVer {DISabled|ENABled}
:DISPlay:SSAVer:AAFTer <time>
```

These commands let you disable or enable the oscilloscope screen saver, and specify a time before the screen saver turns on.

`<time>` An integer; either 2, 3, 4, 5, 6, 7, or 8. The time value specifies the amount of time, in hours, that must pass before the screen saver will turn on.

**Example**

This example enables the oscilloscope screen saver and turns it on in 4 hours.

```
10  OUTPUT 707;":DISPLAY:SSAVER ENABLED"
20  OUTPUT 707;":DISPLAY:SSAVER:AAFT 4"
30  END
```

**Queries**

```
:DISPlay:SSAVer?
:DISPlay:SSAVer:AAFTer?
```

The :DISPlay:SSAVer? and :DISPlay:SSAVer:AAFTer? queries return the state of the screen saver.

**Returned Format**

```
[:DISPlay:SSAVer] {DISabled|ENABled}<NL>
[:DISPlay:SSAVer:AAFTer <time>]<NL>
```

# STRing

Command

```
:DISPlay:STRing "<string_argument>"
```

The :DISPlay:STRing command writes text to the oscilloscope screen. The text is written starting at the current row and column settings. If the column limit is reached (81), the excess text is discarded. The :DISPlay:STRing command does not increment the row value, but :DISPlay:LINE does.

`<string_argument>` Any series of ASCII characters enclosed in quotation marks.

**Example**

This example writes the message "Example 1" to the oscilloscope's display starting at the current row and column settings.

```
10  OUTPUT 707;":DISPLAY:STRING ""Example 1"""
20  END
```

# TEXT

**Command**

```
:DISPlay:TEXT BLANk
```

The :DISPlay:TEXT command blanks the user text area of the screen. This area includes rows 0 through 27, and columns 0 through 81.

**Example**

This example blanks the user text area of the oscilloscope's screen.

```
10  OUTPUT 707;":DISPLAY:TEXT BLANK"
20  END
```

15

# External Channel Commands

# External Channel Commands

The EXTernal channel subsystem commands control the vertical, Y axis functions of the oscilloscope's external channel. These EXTernal commands and queries are implemented in the Infiniium Oscilloscopes:

- BWLimit
- INPut
- PROBe
- PROBe:ATTenuation (only for the 1154A probe)
- PROBe:EADapter
- PROBe:ECoupling
- PROBe:EGAin
- PROBe:GAIN (only for the 1154A probe)
- PROBe:ID?
- PROBe:EOFFset
- PROBe:SKEW
- RANGe
- UNITs

The EXTernal commands only apply to the 54810A and 54820A Infiniium Oscilloscopes.

## BWLimit

Command

```
:EXTernal:BWLimit {{ON|1} | {OFF|0}}
```

The :EXTernal:BWLimit command controls the low-pass filter. When ON, the bandwidth of the external channel is limited. The bandwidth limit filter can be used with either AC or DC coupling.

**Example**

This example sets the internal low-pass filter to "ON" for the external channel.

```
10 OUTPUT 707;":EXTERNAL:BWLIMIT ON"
20 END
```

Query

```
:EXTernal:BWLimit?
```

The :EXTernal:BWLimit? query returns the state of the low-pass filter for the external channel.

Returned Format

```
[:EXTernal:BWLimit] {1|0}<NL>
```

**Example**

This example places the current setting of the low-pass filter in the variable Limit, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":EXTERNAL:BWLIMIT?"
30  ENTER 707;Limit
40  PRINT Limit
50  END
```

# INPut

Command

```
:EXTernal:INPut <parameter>
```

The :EXTernal:INPut command selects the input coupling, impedance, and LF/HF reject for the external channel. The coupling can be set to AC, DC, DC50 or DCFifty, or LFR1 or LFR2 (low-frequency reject).

LFR1 and LFR2 only apply if an 1153A probe is connected to the oscilloscope's External Trigger input. With an 1152A probe attached to the External Trigger input, the :EXTernal:INPut command will not change either the coupling or impedance.

`<parameter>` The parameters available in this command for Infiniium are listed below.

- DC: dc coupling, 1 MΩ input impedance
- DC50 | DCFifty: dc coupling, 50Ω input impedance
- AC: ac 1 MΩ input impedance
- LFR1 | LFR2: ac 1 MΩ input impedance

**Example**

This example sets the external channel input to DC50.

```
10 OUTPUT 707;":EXTERNAL:INPUT DC50"
20 END
```

Query

```
:EXTernal:INPut?
```

The :EXTernal:INPut? query returns the state of the external channel input.

Returned Format

```
[EXTernal:INPut]<parameter><NL>
```

**Example**

This example places the current input for the external channel in the string variable, Input$. The program then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":EXTERNAL:INPUT?
30  ENTER 707;Input$
40  PRINT Input$
50  END
```

# PROBe

**Command**

```
:EXTernal:PROBe {<attenuation_factor>,{RATio | DECibel}}
```

The :EXTernal:PROBe command sets the probe attenuation factor and, optionally, the units for the probe attenuation factor. The range of the probe attenuation factor is from 0.0001 to 1,000,000 and from -80 dB to 120 dB. The reference factors that are used for scaling the display are changed with this command, and affect automatic measurements and trigger levels.

`<attenuation_factor>` A real number from 0.0001 to 1,000,000, and -80 dB to 120 dB, representing the probe attenuation factor; the factor depends on the units.

**Example**

This example sets the probe attenuation factor of the external channel to 10, and the units to decibel.

```
10  OUTPUT 707;":EXTERNAL:PROBE 10,DEC"
20  END
```

**Query**

```
:EXTernal:PROBe?
```

The :EXTernal:PROBe? query returns the current probe attenuation setting for the external channel and the units.

**Returned Format**

```
[:EXTernal:PROBe] <attenuation_factor>,{RATio | DECibel}<NL>
```

**Example**

This example places the current attenuation setting for the external channel in the string variable, Atten$, and prints the contents.

```
10  DIM Atten$[50]!Dimension variable
20  OUTPUT 707;":EXTERNAL:PROBE?"
30  ENTER 707;Atten$
40  PRINT Atten$
50  END
```

# PROBe:ATTenuation

Command

```
:EXTernal:PROBe:ATTenuation {DIV1 | DIV10}
```

The :EXTernal:PROBe:ATTenuation command sets the probe's attenuation. There are some Infiniium active and differential probes that have the ability to change the probe's input amplifier's attenuation.

This command is only available when an Infiniium active or differential probe is connected to the channel. If one of these probes is not connected to the external channel you will get a settings conflict error.

**Example**

This example sets the probe attenuation to divide by 10.

```
10  OUTPUT 707;":EXTERNAL:PROBE:ATTENUATION DIV10"
20  END
```

Query

```
:EXTernal:PROBe:ATTenuation?
```

The :EXTernal:PROBe:ATTenuation? query returns the current probe attenuation setting.

Returned Format

```
[:EXTernal:PROBe:ATTenuation] {DIV1 | DIV10}<NL>
```

# PROBe:EADapter

Command

```
:EXTernal:PROBe:EADapter {NONE | AC | DIV10 |
DIV20 | DIV100}
```

The :EXTernal:EADapter command sets the Infiniium external adapter control. There are some probes that have external adapters that you can attach to the end of your probe. When you attach one of these adapters, you should use the EADapter command to set the external adapter control to match the adapter connected to your probe as follows.

| Parameter | Description |
|---|---|
| NONE | Use this setting when there is no adapter connected to the end of your probe. |
| AC | Use this setting when you have an ac coupling adapter connected to the end of your probe. |
| DIV10 | Use this setting when you have a divide by 10 adapter connected to the end of your probe. |
| DIV20 | Use this setting when you have a divide by 20 adapter connected to the end of your probe. |
| DIV100 | Use this setting when you have a divide by 100 adapter connected to the end of your probe. |

**Example**

This example sets the external adapter to divide by 10:

```
10  OUTPUT 707;":EXTERNAL:PROBE:EADAPTER DIV10"
20  END
```

Query

```
:EXTernal:PROBe:EADapter?
```

The :EXTernal:PROBe:EADapter? query returns the external adapter value.

Returned Format

```
[CHANnel<N>:EDApter] {NONE | AC | DIV10 | DIV20 |
DIV100}<NL>
```

**Example**

This example places the external adapter value in the string variable, Adapter$, then prints the contents of the variable to the computer's screen.

```
10  DIM Adapter$[50]!Dimension variable
20  OUTPUT 707;":EXTERNAL:EADAPTER?
30  ENTER 707;Adapter$
40  PRINT Adapter$
50  END
```

# PROBe:ECoupling

Command

```
:EXTernal:PROBe:ECoupling {NONE | AC}
```

The :EXTernal:PROBe:ECoupling command sets the Infiniium external coupling adapter control. There are some probes that have external coupling adapters that you can attach to the end of your probe. When you attach one of these adapters, you should use the ECoupling command to set the external coupling adapter control to match the adapter connected to your probe as follows.

| Parameter | Description |
|---|---|
| NONE | Use this setting when there is no adapter connected to the end of your probe. |
| AC | Use this setting when you have an ac coupling adapter connected to the end of your probe. |

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**Example**

This example sets the external coupling adapter for external trigger channel to ac:

```
10  OUTPUT 707;":EXTERNAL:PROBE:ECOUPLING AC"
20  END
```

Query

```
:EXTernal:PROBe:ECoupling?
```

The :EXTernal:PROBe:ECoupling? query returns the current external coupling adapter value for the external trigger channel.

Returned Format

```
[EXTernal:PROBe:ECoupling] {NONE | AC}<NL>
```

**Example**

This example places the external coupling adapter value of the external trigger channel in the string variable, Adapter$, then prints the contents of the variable to the computer's screen.

```
10  DIM Adapter$[50]!Dimension variable
20  OUTPUT 707;":EXTERNAL:PROBE:ECOUPLING?
30  ENTER 707;Adapter$
40  PRINT Adapter$
50  END
```

# PROBe:EGAin

**Command**

```
:EXTernal:PROBe:EGAin <gain_value>
```

The :EXTernal:PROBe:EGAin command sets the probe gain. The units of volts, amperes, watts, and unknown are set using the :EXTernal:UNITs command.

`<gain_value>` A real number for the gain value.

**Example**

This example sets the probe gain for the external channel to 125 x $10^{-3}$.

```
10  OUTPUT 707;":EXTERNAL:PROBE:EGAIN 125E-3"
20  END
```

**Query**

```
:EXTernal:PROBe:EGAin?
```

The :EXTernal:PROBe:EGAin? query returns the gain setting for the external channel.

**Returned Format**

```
[:EXTernal:PROBe:EGAin] <gain_value><NL>
```

# PROBe:EOFFset

Command

```
:EXTernal:PROBe:EOFFset <offset_value>
```

The :EXTernal:PROBe:EOFFset command sets the probe offset. The units of volts, amperes, watts, and unknown are set using the :EXTernal:UNITs command.

`<offset_value>` A real number for the offset value.

**Example**

This example sets the probe offset for the external channel to 125 x $10^{-3}$.

```
10  OUTPUT 707;":EXTERNAL:PROBE:EOFFSET 125E-3"
20  END
```

Query

```
:EXTernal:PROBe:EOFFset?
```

The :EXTernal:PROBe:EOFFset? query returns the offset value.

Returned Format

```
[:EXTernal:PROBe:EOFFset] <offset_value><NL>
```

# PROBe:GAIN

Command

```
:EXTernal:PROBe:GAIN {X1 | X10}
```

The :EXTernal:PROBe:GAIN command sets the probe gain. There are some Infiniium active and differential probes that have the ability to change the probe's input amplifier gain.

This command is only available when an Infiniium active or differential probe is connected to the external channel. If one of these probes is not connected to the external channel you will get a settings conflict error.

The units of volts, amperes, watts, and unknown are set using the :EXTernal:UNITs command.

**Example**

This example sets the probe gain to times 10.

```
10  OUTPUT 707;":EXTERNAL:PROBE:GAIN X10"
20  END
```

Query

```
:EXTernal:PROBe:GAIN?
```

The :EXTernal:PROBe:GAIN? query returns the probe gain setting.

Returned Format

```
[:EXTernal:PROBe:GAIN] {X1 | X10}<NL>
```

# PROBe:ID?

Query

```
:EXTernal:PROBe:ID?
```

The :EXTernal:PROBe:ID? query returns the type of probe attached to the Oscilloscope.

**Example**

This example reports the probe type connected to channel 1, if one is connected.

```
10  OUTPUT 707;":EXTernal:PROBE:ID?"
20  END
```

# PROBe:SKEW

Command

```
:EXTernal:PROBe:SKEW <skew_value>
```

The :EXTernal:PROBe:SKEW command sets the value of the External Trigger probe skew.

`<skew_value>` A real number from -100E-6 to 100E-6.

**Example**

This example sets the external probe skew to 10 microseconds.

```
10  OUTPUT 707;":EXTERNAL:PROBE:SKEW 10E-6"
20  END
```

Query

```
:EXTernal:PROBe:SKEW?
```

The :EXTernal:PROBe:SKEW? query returns the current skew setting for the external channel.

Returned Format

```
[:EXTernal:PROBe:SKEW] <skew_value><NL>
```

**See Also**

For information on skew, see the Calibration Commands chapter.

# RANGe

Command

```
:EXTernal:RANGe <range_value>
```

The :EXTernal:RANGe command defines the vertical axis of the external channel. The value represents the full-scale deflection of the vertical axis in volts. This value changes as the probe attenuation factor is changed. If you change the probe attenuation, the range value is multiplied by the probe attenuation factor.

`<range_value>` Voltage setting of 1, 5, or 25, corresponding to ±1V, ±5V, or ±25V for the external channel vertical range.

**Example**

This example sets the vertical range for the external channel to ±5V.

```
10  OUTPUT 707;":EXTERNAL:RANGE 5"
20  END
```

Query

```
:EXTernal:RANGe?
```

The :EXTernal:RANGe? query returns the current vertical axis setting for the external channel.

Returned Format

```
[:EXTernal:RANGe]<range value><NL>
```

**Example**

This example places the current range value in the number variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":EXTERNAL:RANGE?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

# UNITs

Command

```
:EXTernal:UNITs {VOLT | AMPere | WATT | UNKNown}
```

The :EXTernal:UNITs command sets the vertical units. You can specify Y-axis units of VOLTS, AMPS, WATTs, or UNKNown. The units are implied for other pertinent channel commands (such as RANGe and OFFSet). See the Probe Setup dialog box for more information.

**Example**

This example sets the units for the external channel to amperes.

```
10  OUTPUT 707;":EXTERNAL:UNITS AMPERE"
20  END
```

Query

```
:EXTernal:UNITs?
```

The :EXTernal:UNITs? query returns the current units setting for the external channel.

Returned Format

```
[:EXTernal:UNITs] {VOLT | AMPere | WATT | UNKNown}<NL>
```

**Example**

This example places the vertical units for the external channel in the string variable, Units$, then prints the contents of the variable to the computer's screen.

```
10 DIM Units$[50]
20 OUTPUT 707;"EXTERNAL:UNITS?"
30 ENTER 707;Units$
40 PRINT Units$
50 END
```

16

# Function Commands

# Function Commands

The FUNCtion subsystem defines functions 1 - 4. The operands of these functions can be any of the installed channels in the oscilloscope, waveform memories 1 - 4, functions 1 - 4, or a constant. These FUNCtion commands and queries are implemented in the Infiniium Oscilloscopes:

- FUNCtion<N>?
- ADD
- AVERage
- DIFF (Differentiate)
- DISPlay
- DIVide
- FFT:FREQuency
- FFT:RESolution?
- FFT:WINDow
- FFTMagnitude
- HORizontal
- HORizontal:POSition
- HORizontal:RANGe
- INTegrate
- INVert
- MAGNify
- MAXimum
- MEASurement (Only available on the 54845A and 54846A)
- MINimum
- MULTiply
- OFFSet
- RANGe
- SUBTract
- VERSus
- VERTical
- VERTical:OFFset

- VERTical:RANGe

You can control the vertical scaling and offset functions remotely using the RANGe and OFFSet commands in this subsystem. You can obtain the horizontal scaling and position values of the functions using the :HORizontal:RANge? and :HORizontal:POSition? queries in this subsystem.

If a channel is not on but is used as an operand, that channel will acquire waveform data.

If the operand waveforms have different memory depths, the function uses the shorter of the two.

If the two operands have the same time scales, the resulting function has the same time scale. If the operands have different time scales, the resulting function has no valid time scale. This is because operations are performed based on the displayed waveform data position, and the time relationship of the data records cannot be considered. When the time scale is not valid, delta time pulse parameter measurements have no meaning, and the unknown result indicator is displayed on the screen.

Constant operands take on the same time scale as the associated waveform operand.

# FUNCtion<N>?

Query

```
:FUNCtion<N>?
```

The :FUNCtion<N>? query returns the currently defined source(s) for the function.

Returned Format

```
[:FUNCtion<N>:<operator>] {<operand>,[,<operand>]}<NL>
```

<N> An integer, 1 - 4, representing the selected function.

<operator> Active math operation for the selected function: ADD, AVERage, DIFF, DIVide, FFTMagnitude, INTegrate, INVert, MAGNify, MAXimum, MINimum, MULTiply, SUBTract, or VERSus.

<operand> Any allowable source for the selected FUNCtion, including channels, waveform memories 1-4, and functions 1-4. If the function is applied to a constant, the source returns the constant.

The channel number is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**Example**

This example returns the currently defined source for function 1.

```
10  OUTPUT 707;":FUNCTION1?"
20  END
```

If the headers are off (see :SYSTem:HEADer), the query returns only the operands, not the operator.

```
10  :SYST:HEAD ON
20  :FUNC1:ADD CHAN1,CHAN2
30  :FUNC1?  !returns :FUNC1:ADD CHAN1,CHAN2
40  :SYST:HEAD OFF
50  :FUNC1?  !returns CHAN1,CHAN2
```

# ADD

Command

```
:FUNCtion<N>:ADD <operand>,<operand>
```

The :FUNCtion<N>:ADD command defines a function that takes the algebraic sum of the two operands.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6.

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

**Example**

This example sets up function 1 to add channel 1 to channel 2.

```
10  OUTPUT 707;":FUNCTION1:ADD CHANNEL1,CHANNEL2"
20  END
```

# AVERage

Command

```
:FUNCtion<N>:AVERage <operand>[,<averages>]
```

The :FUNCtion<N>:AVERage command defines a function that averages the operand based on the number of specified averages.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6

<avereages> An integer, 2 to 4096 specifing the number of waveforms to be averaged

**Example**

This example sets up function 1 to average channel 1 using 16 averages.

```
10  OUTPUT 707;":FUNCTION1:AVERAGE CHANNEL1,16"
20  END
```

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

# DIFF (Differentiate)

Command

```
:FUNCtion<N>:DIFF <operand>
```

The :FUNCtion<N>:DIFF command defines a function that computes the discrete derivative of the operand.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6.

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

**Example**

This example sets up function 2 to take the discrete derivative of the waveform on channel 2.

```
10  OUTPUT 707;":FUNCTION2:DIFF CHANNEL2"
20  END
```

# DISPlay

Command

```
:FUNCtion<N>:DISPlay {{ON|1} | {OFF|0}}
```

The :FUNCtion<N>:DISPlay command either displays the selected function or removes it from the display.

<N> An integer, 1 - 4, representing the selected function.

**Example**

This example turns function 1 on.

```
10  OUTPUT 707;":FUNCTION1:DISPLAY ON"
20  END
```

Query

```
:FUNCtion<N>:DISPlay?
```

The :FUNCtion<N>:DISPlay? query returns the displayed status of the specified function.

Returned Format

```
[:FUNCtion<N>:DISPlay] {1|0}<NL>
```

**Example**

This example places the current state of function 1 in the variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":FUNCTION1:DISPLAY?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

# DIVide

Command

```
:FUNCtion<N>:DIVide <operand>,<operand>
```

The :FUNCtion<N>:DIVide command defines a function that divides the first operand by the second operand.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6.

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

**Example**

This example sets up function 2 to divide the waveform on channel 1 by the waveform in waveform memory 4.

```
10  OUTPUT 707;":FUNCTION2:DIVIDE CHANNEL1,WMEMORY4"
20  END
```

# FFT:FREQuency

**Command**

```
:FUNCtion<N>:FFT:FREQuency <center_frequency_value>
```

The :FUNCtion<N>:FFT:FREQuency command sets the center frequency for the FFT when :FUNCtion<N>:FFTMagnitude is defined for the selected function.

`<N>` An integer, 1 - 4, representing the selected function.

`<center_frequency_value>` A real number for the value in Hertz, from -1E12 to 4E9.

**Query**

```
:FUNCtion<N>:FFT:FREQuency?
```

The :FUNCtion<N>:FFT:FREQuency? query returns the center frequency value.

**Returned Format**

```
[FUNCtion<N>:FFT:FREQuency] <center_frequency_value><NL>
```

## FFT:RESolution?

**Query**

```
:FUNCtion<N>:FFT:RESolution?
```

The :FUNCtion<N>:FFT:RESolution? query returns the current resolution of the FFT function.

**Returned Format**

```
[FUNCtion<N>:FFT:RESolution] <resolution_value><NL>
```

`<N>` An integer from 1 to 4 representing the selected function.

`<resolution_value>` Resolution frequency.

The FFT resolution is determined by the sample rate and memory depth settings. The FFT resolution is calculated using the following equation:

FFT Resolution = Sample Rate / Effective Memory Depth

The effective memory depth is the highest power of 2 less than or equal to the number of sample points across the display. The memory bar in the status area at the top of the display indicates how much of the actual memory depth is across the display.

# FFT:WINDow

Command

```
:FUNCtion<N>:FFT:WINDow {RECTangular | HANNing |
FLATtop}
```

The :FUNCtion<N>:FFT:WINDow command sets the window type for the FFT function.

The FFT function assumes that the time record repeats. Unless there is an integral number of cycles of the sampled waveform in the record, a discontinuity is created at the beginning of the record. This introduces additional frequency components into the spectrum about the actual peaks, which is referred to as spectral leakage. To minimize spectral leakage, windows that approach zero smoothly at the beginning and end of the record are employed as filters to the FFTs. Each window is useful for certain classes of input waveforms.

- The RECTangular window is essentially no window, and all points are multiplied by 1. This window is useful for transient waveforms and waveforms where there are an integral number of cycles in the time record.
- The HANNing window is useful for frequency resolution and general purpose use. It is good for resolving two frequencies that are close together, or for making frequency measurements.
- The FLATtop window is best for making accurate amplitude measurements of frequency peaks.

<N> An integer, 1 - 4, representing the selected function. This command presently selects all functions, regardless of which integer (1-4) is passed.

**Example**

This example sets the window type for the FFT function to RECTangular.

```
10 OUTPUT 707;":FUNCTION<N>:FFT:WINDOW RECTANGULAR
20 END
```

Query

```
:FUNCtion<N>:FFT:WINDow?
```

The :FUNCtion<N>:FFT:WINDow? query returns the current selected window for the FFT function.

Returned Format

```
[:FUNCtion<N>:FFT:WINDow] {RECTangular | HANNing |
FLATtop}<NL>
```

**Example**

This example places the current state of the function 1 FFT window in the string variable, WND?, then prints the contents of the variable to the computer's screen.

```
10 DIM WND$[50]
20 OUTPUT 707;":FUNCTION1:FFT:WINDOW?"
30 ENTER 707;WND$
40 PRINT WND$
50 END
```

# FFTMagnitude

Command

```
:FUNCtion<N>:FFTMagnitude <operand>
```

The :FUNCtion<N>:FFTMagnitude command computes the Fast Fourier Transform (FFT) of the specified channel, function, or memory. The FFT takes the digitized time record and transforms it to magnitude and phase components as a function of frequency.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6.

**Example**

This example sets up function 1 to compute the FFT of waveform memory 3.

```
10 OUTPUT 707;":FUNCTION1:FFTMAGNITUDE WMEMORY3"
20 END
```

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

# FFTPhase

Command

```
:FUNCtion<N>:FFTPhase <source>
```

The :FUNCtion<N>:FFTPhase command computes the Fast Fourier Transform (FFT) of the specified channel, function, or waveform memory. The FFT takes the digitized time record and transforms it into magnitude and phase components as a function of frequency.

`<N>` An integer, 1 - 4, representing the selected function.

`<source>` {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6.

**Example**

This example sets up function 1 to compute the FFT of waveform memory 3.

```
10 OUTPUT 707;":FUNCTION1:FFTPHASE WMEMORY3"
20 END
```

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

# HORizontal

**Command**

```
:FUNCtion<N>:HORizontal {AUTO | MANual}
```

The :FUNCtion<N>:HORizontal command sets the horizontal tracking to either AUTO or MANual.

The HORizontal command also includes the following commands and queries, which are described on the following pages:

- POSition
- RANGe

`<N>` An integer, 1 - 4, representing the selected function.

**Query**

```
:FUNCtion<N>:HORizontal?
```

The :FUNCtion<N>:HORizontal? query returns the current horizontal scaling mode of the specified function.

**Returned Format**

```
[:FUNCtion<N>:HORizontal] {AUTO | MANual}<NL>
```

**Example**

This example places the current state of the function 1 horizontal tracking in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":FUNCTION1:HORIZONTAL?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# HORizontal:POSition

**Command**

```
:FUNCtion<N>:HORizontal:POSition <position_value>
```

The :FUNCtion<N>:HORizontal:POSition command sets the time value at center screen for the selected function. If the oscilloscope is not already in manual mode when you execute this command, it puts the oscilloscope in manual mode.

When you select :FUNCtion<N>:FFTMagnitude, the horizontal position is equivalent to the center frequency. This also automatically selects manual mode.

`<N>` An integer, 1 - 4, representing the selected function.

`<position_value>` A real number for the position value in time, in seconds, from -1E12 to 4E9.

**Query**

```
:FUNCtion<N>:HORizontal:POSition?
```

The :FUNCtion<N>:HORizontal:POSition? query returns the current time value at center screen of the selected function.

**Returned Format**

```
[:FUNCtion<N>:HORizontal:POSition] <position><NL>
```

**Example**

This example places the current horizontal position setting for function 2 in the numeric variable, Value, then prints the contents to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":FUNCTION2:HORIZONTAL:POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# HORizontal:RANGe

**Command**

```
:FUNCtion<N>:HORizontal:RANGe <range_value>
```

The :FUNCtion<N>:HORizontal:RANGe command sets the current time range for the specified function. This automatically selects manual mode.

`<N>` An integer, 1 - 4, representing the selected function.

`<range_value>` A real number for the width of screen in current X-axis units (usually seconds), from 1E-12 to 5E12.

**Query**

```
:FUNCtion<N>:HORizontal:RANGe?
```

The :FUNCtion<N>:HORizontal:RANGe? query returns the current time range setting of the specified function.

**Returned Format**

```
[:FUNCtion<N>:HORizontal:RANGe] <range><NL>
```

**Example**

This example places the current horizontal range setting of function 2 in the numeric variable, Value, then prints the contents to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":FUNCTION2:HORIZONTAL:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# INTegrate

Command

```
:FUNCtion<N>:INTegrate <operand>
```

The :FUNCtion<N>:INTegrate command defines a function that computes the integral of the specified operand's waveform.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6.

**Example**

This example sets up function 1 to compute the integral of waveform memory 3.

```
10  OUTPUT 707;":FUNCTION1:INTEGRATE WMEMORY3"
20  END
```

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

# INVert

Command

```
:FUNCtion<N>:INVert <operand>
```

The :FUNCtion<N>:INVert command defines a function that inverts the defined operand's waveform by multiplying by -1.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6.

**Example**

This example sets up function 2 to invert the waveform on channel 1.

```
10  OUTPUT 707;":FUNCTION2:INVERT CHANNEL1"
20  END
```

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

# MAGNify

Command

```
:FUNCtion<N>:MAGNify <operand>
```

The :FUNCtion<N>:MAGNify command defines a function that is a copy of the operand. The magnify function is a software magnify. No hardware settings are altered as a result of using this function. It is useful for scaling channels, another function, or memories with the RANGe and OFFSet commands in this subsystem.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6.

**Example**

This example creates a function (function 1) that is a magnified version of channel 1.

```
10  OUTPUT 707;":FUNCTION1:MAGNIFY CHANNEL1"
20  END
```

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

# MAXimum

Command

```
:FUNCtion<N>:MAXimum <operand>
```

The :FUNCtion<N>:MAXimum command defines a function that computes the maximum value of the operand waveform in each time bucket.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6.

**Example**

This example sets up function 2 to compute the maximum of each time bucket for channel 2.

```
10  OUTPUT 707;":FUNCTION2:MAXIMUM CHANNEL2"
20  END
```

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

## MEASurement

Command

```
:FUNCtion<N>:MEASurement {MEAS1 | MEAS2 | MEAS3 |
MEAS4}
```

The :FUNCtion<N>:MEASurement command defines a function that creates a graph of the selected measurement versus time. Only one function that uses the measurement operator can be display at a time. If you try to turn more than one on, an error is generated.

This command is only available on the 54845A and 54846A oscilloscopes.

The measurement is computed for every cycle of the waveform that is in the waveform viewing area. This command returns an error when there are no automated timing measurements on. The measurements that are allowed for this command are:

- + width
- - width
- Duty Cycle
- Fall Time
- Frequency
- Period
- Rise Time

<N> An integer, 1 - 4, representing the selected function.

**Example**

This example defines function 1 to have a measurement operator which graphs measurement 2.

```
10  OUTPUT 707;":FUNCTION1:MEASUREMENT MEAS2"
20  END
```

Query

```
:FUNCtion<N>:MEASurement?
```

The :FUNCtion<N>:MEASurement? query returns the measurement being used by the measurement operator.

If no valid measurements are on, an empty string is returned.

This query is only available on the 54845A and 54846A oscilloscopes.

Returned Format

```
[:FUNCtion<N>:MEASurement] {MEAS1 | MEAS2 | MEAS3 |
MEAS4}<NL>
```

**Example**

This example places the measurement being used by function 1 in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":FUNCTION1:MEASUREMENT?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# MINimum

Command

```
:FUNCtion<N>:MINimum <operand>
```

The :FUNCtion<N>:MINimum command defines a function that computes the minimum of each time bucket for the defined operand's waveform.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6.

**Example**

This example sets up function 2 to compute the minimum of each time bucket for channel 4.

```
10  OUTPUT 707;":FUNCTION2:MINIMUM CHANNEL4"
20  END
```

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

# MULTiply

Command

```
:FUNCtion<N>:MULTiply <operand>,<operand>
```

The :FUNCtion<N>:MULTiply command defines a function that algebraically multiplies the first operand by the second operand.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6.

**Example**

This example defines a function that multiplies channel 1 by waveform memory 1.

```
10  OUTPUT 707;":FUNCTION1:MULTIPLY CHANNEL1,WMEMORY1"
20  END
```

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

# OFFSet

Command

```
:FUNCtion<N>:OFFSet <offset_value>
```

The :FUNCtion<N>:OFFSet command sets the voltage represented at the center of the screen for the selected function. This automatically changes the mode from auto to manual.

`<N>` An integer, 1 - 4, representing the selected function.

`<offset_value>` A real number for the vertical offset in the currently selected Y-axis units (normally volts). The offset value is limited to being within the vertical range that can be represented by the function data.

**Example**

This example sets the offset voltage for function 1 to 2 mV.

```
10  OUTPUT 707;":FUNCTION1:OFFSET 2E-3"
20  END
```

Query

```
:FUNCtion<N>:OFFSet?
```

The :FUNCtion<N>:OFFSet? query returns the current offset value for the selected function.

Returned Format

```
[:FUNCtion<N>:OFFSet] <offset_value><NL>
```

**Example**

This example places the current setting for offset on function 2 in the numeric variable, Value, then prints the result to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":FUNCTION2:OFFSET?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# RANGe

Command

```
:FUNCtion<N>:RANGe <full_scale_range>
```

The :FUNCtion<N>:RANGe command defines the full-scale vertical axis of the selected function. This automatically changes the mode from auto to manual.

`<N>` An integer, 1 - 4, representing the selected function.

`<full_scale_range>` A real number for the full-scale vertical range, from 10E-18 to 1E15.

**Example**

This example sets the full-scale range for function 1 to 400 mV.

```
10  OUTPUT 707;":FUNCTION1:RANGE 400E-3"
20  END
```

Query

```
:FUNCtion<N>:RANGe?
```

The :FUNCtion<N>:RANGe? query returns the current full-scale range setting for the specified function.

Returned Format

```
[:FUNCtion<N>:RANGe] <full_scale_range><NL>
```

**Example**

This example places the current range setting for function 2 in the numeric variable "Value," then prints the contents to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":FUNCTION2:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# SUBTract

**Command**

```
:FUNCtion<N>:SUBTract <operand>,<operand>
```

The :FUNCtion<N>:SUBTract command defines a function that algebraically subtracts the second operand from the first operand.

`<N>` An integer, 1 - 4, representing the selected function.

`<operand>` {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6.

**Example**

This example defines a function that subtracts waveform memory 1 from channel 1.

```
10 OUTPUT 707;":FUNCTION1:SUBTRACT CHANNEL1,WMEMORY1"
20 END
```

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

# VERSus

Command

```
:FUNCtion<N>:VERSus <operand>,<operand>
```

The :FUNCtion<N>:VERSus command defines a function for an X-versus-Y display. The first operand defines the Y axis and the second defines the X axis. The Y-axis range and offset are initially equal to that of the first operand, and you can adjust them with the RANGe and OFFSet commands in this subsystem.

<N> An integer, 1 - 4, representing the selected function.

<operand> {CHANnel<n> | FUNCtion<n> | WMEMory<n> | <float_value>}

CHANnel<n> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<n> and WMEMory<n> are:

An integer, 1 - 4, representing the selected function or waveform memory.

<float_value> is:

A real number from -1E6 to 1E6.

**Example**

This example defines function 1 as an X-versus-Y display. Channel 1 is the X axis and waveform memory 2 is the Y axis.

```
10  OUTPUT 707;":FUNCTION1:VERSUS WMEMORY2,CHANNEL1"
20  END
```

**Functions Used as Operands**

A function may be used as a source for another function, subject to the following constraints:

F4 can have F1, F2, or F3 as a source.

F3 can have F1 or F2 as a source.

F2 can have F1 as a source.

F1 cannot have any other function as a source.

# VERTical

Command

```
:FUNCtion<N>:VERTical {AUTO | MANual}
```

The :FUNCtion<N>:VERTical command sets the vertical scaling mode of the specified function to either AUTO or MANual.

This command also contains the following commands and queries:

- OFFset
- RANge

<N> An integer, 1 - 4, representing the selected function.

Query

```
:FUNCtion<N>:VERTical?
```

The :FUNCtion<N>:VERTical? query returns the current vertical scaling mode of the specified function.

Returned Format

```
[:FUNCtion<N>:VERTical] {AUTO | MANual}<NL>
```

**Example**

This example places the current state of the vertical tracking of function 1 in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":FUNCTION1:VERTICAL?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# VERTical:OFFSet

**Command**

```
:FUNCtion<N>:VERTical:OFFSet <offset_value>
```

The :FUNCtion<N>:VERTical:OFFSet command sets the voltage represented at center screen for the selected function. This automatically changes the mode from auto to manual.

`<N>` An integer, 1 - 4, representing the selected function.

`<offset_value>` A real number for the vertical offset in the currently selected Y-axis units (normally volts). The offset value is limited only to being within the vertical range that can be represented by the function data.

**Query**

```
:FUNCtion<N>:VERTical:OFFset?
```

The :FUNCtion<N>:VERTical:OFFSet? query returns the current offset value of the selected function.

**Returned Format**

```
[:FUNCtion<N>:VERTical:OFFset] <offset_value><NL>
```

**Example**

This example places the current offset setting for function 2 in the numeric variable, Value, then prints the contents to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":FUNCTION2:VERTICAL:OFFSET?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

## VERTical:RANGe

**Command**

```
:FUNCtion<N>:VERTical:RANGe <full_scale_range>
```

The :FUNCtion<N>:VERTical:RANGe command defines the full-scale vertical axis of the selected function. This automatically changes the mode from auto to manual, if the oscilloscope is not already in manual mode.

`<N>` An integer, 1 - 4, representing the selected function.

`<full_scale_range>` A real number for the full-scale vertical range, from 10E-18 to 1E5.

**Query**

```
:FUNCtion<N>:VERTical:RANGe?
```

The :FUNCtion<N>:VERTical:RANGe? query returns the current range setting of the specified function.

**Returned Format**

```
[:FUNCtion<N>:VERTical:RANGe] <range><NL>
```

**Example**

This example places the current vertical range setting of function 2 in the numeric variable, Value, then prints the contents to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":FUNCTION2:VERTICAL:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

17

# Hardcopy Commands

# Hardcopy Commands

The HARDcopy subsystem commands set various parameters for printing the screen. The print sequence is activated when the root level command :PRINt is sent.

These HARDcopy commands and queries are implemented in the Infiniium Oscilloscopes:

- AREA
- DPRinter
- FACTors
- IMAGe
- PRINTers?

# AREA

Command

```
:HARDcopy:AREA {GRATicule | SCReen}
```

The :HARDcopy:AREA command selects which data from the screen is to be printed. When you select GRATicule, only the graticule area of the screen is printed (this is the same as choosing Waveforms Only in the Configure Printer dialog box). When you select SCReen, the entire screen is printed.

**Example**

This example selects the graticule for printing.

```
10  OUTPUT 707;":HARDCOPY:AREA GRATICULE"
20  END
```

Query

```
:HARDcopy:AREA?
```

The :HARDcopy:AREA? query returns the current setting for the area of the screen to be printed.

Returned Format

```
[:HARDcopy:AREA] {GRATicule | SCReen}<NL>
```

**Example**

This example places the current selection for the area to be printed in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10  DIM Selection$[50]!Dimension variable
20  OUTPUT 707;":HARDCOPY:AREA?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

# DPRinter

Command

```
:HARDcopy:DPRinter
{<printer_number>|<printer_string>}
```

The :HARDcopy:DPRinter command selects the default printer to be used.

`<printer_number>` An integer representing the attached printer. This number corresponds to the number returned with each printer name by the :HARDcopy:PRINters? query.

`<printer_string>` A string of alphanumeric characters representing the attached printer.

The :HARDcopy:DPRinter command specifies a number or string for the printer attached to the oscilloscope. The printer string must exactly match the character strings in the File->Print Setup dialog boxes, or the strings returned by the :HARDcopy:PRINters? query.

**Examples**

This example sets the default printer to the second installed printer returned by the :HARDcopy:PRINters? query.

```
10  OUTPUT 707;":HARDCOPY:DPRINTER 2"
20  END
```

This example sets the default printer to the installed printer with the name "HP Laser".

```
10  OUTPUT 707;":HARDCOPY:DPRINTER ""HP Laser"""
20  END
```

Query

```
:HARDcopy:DPRinter?
```

The :HARDcopy:DPRinter? query returns the current printer number and string.

Returned Format

```
[:HARDcopy:DPRinter?]
{<printer_number>,<printer_string>,DEFAULT}<NL>
```

Or, if there is no default printer (no printers are installed), only a <NL> is returned.

**Example**

This example places the current setting for the hard copy printer in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":HARDCOPY:DPRinter?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

**Programs Must Wait After Changing the Default Printer**

It takes several seconds to change the default printer. Any programs that try to set the default printer must wait (10 seconds is a safe amount of time) for the change to complete before sending other commands. Otherwise the oscilloscope will become unresponsive.

# FACTors

Command

```
:HARDcopy:FACTors {{ON|1} | {OFF|0}}
```

The :HARDcopy:FACTors command determines whether the oscilloscope setup factors will be appended to screen or graticule images. FACTors ON is the same as choosing Include Setup Information in the Configure Printer dialog box.

**Example**

This example turns on the setup factors.

```
10  OUTPUT 707;":HARDCOPY:FACTORS ON"
20  END
```

Query

```
:HARDcopy:FACTors?
```

The :HARDcopy:FACTors? query returns the current setup factors setting.

Returned Format

```
[:HARDcopy:FACTors] {1|0}<NL>
```

**Example**

This example places the current setting for the setup factors in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":HARDCOPY:FACTORS?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# IMAGe

Command

```
:HARDcopy:IMAGe {NORMal | INVert}
```

The :HARDcopy:IMAGe command prints the image normally, inverted, or in monochrome. IMAGe INVert is the same as choosing Invert Waveform Colors in the Configure Printer dialog box.

**Example**

This example sets the hard copy image output to normal.

```
10  OUTPUT 707;":HARDCOPY:IMAGE NORMAL"
20  END
```

Query

```
:HARDcopy:IMAGe?
```

The :HARDcopy:IMAGe? query returns the current image setting.

Returned Format

```
[:HARDcopy:IMAGe] {NORMal | INVert}<NL>
```

**Example**

This example places the current setting for the hard copy image in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":HARDCOPY:IMAGE?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# PRINters?

Query

```
:HARDcopy:PRINters?
```

The :HARDcopy:PRINters? query returns the currently available printers.

Returned Format

```
[:HARDcopy:PRINters?]
<printer_count><NL><printer_data><NL>[,<printer_data><NL>]
```

<printer_count> The number of printers currently installed.

<printer_data> The printer number and the name of an installed printer. The word DEFAULT appears next to the printer that is the currently selected default printer.

The <printer_data> return string has the following format:
<printer_number>,<printer_string>{,DEFAULT}

**Example**

This example places the number of installed printers into the variable Count, loops through it that number of times, and prints the installed printer names to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":HARDCOPY:PRINTERS?"
30  ENTER 707;Count
40  IF Count>0 THEN
50  FOR Printer_number=1 TO Count
60  ENTER 707;Setting$
70  PRINT Setting$
80  NEXT Printer_number
90  END IF
100 END
```

18

# Histogram Commands

# Histogram Commands

The HISTogram commands and queries control the histogram features. A histogram is a probability distribution that shows the distribution of acquired data within a user-definable histogram window.

You can display the histogram either vertically, for voltage measurements, or horizontally, for timing measurements.

The most common use for histograms is measuring and characterizing noise or jitter on displayed waveforms. Noise is measured by sizing the histogram window to a narrow portion of time and observing a veritcal histogram that measures the noise on a waveform. Jitter is measured by sizing the histogram window to a narrow portion of voltage and observing a horizontal histogram that measures the jitter on an edge.

These HISTogram commands and queries are implemented in the Infiniium Oscilloscopes:

- AXIS
- MODE
- SCALe:SIZE
- WINDow:DEFault
- WINDow:SOURce
- WINDow:X1Position|LLIMit
- WINDow:X2Position|RLIMit
- WINDow:Y1Position|TLIMit
- WINDow:Y2Position|BLIMit

### Histograms and the database

The histograms, mask testing, and color grade persistence use a specific database that uses a different memory area from the waveform record for each channel. When any of these features are turned on, the oscilloscope starts building the database. The database is the size of the graticule area. Behind each pixel is a 21-bit counter that is incremented each time data from a channel or function hits a pixel. The maximum count (saturation) for each counter is 2,097,151. You can use the DISPlay:CGRade:LEVels command to see if any of the counters are close to saturation.

The database continues to build until the oscilloscope stops acquiring data or all three features (color grade persistence, mask testing, and histograms) are turned off. You can clear the database by turning off all three features that use the database.

The database does not differentiate waveforms from different channels or functions. If three channels are on and the waveform from each channel happens to light the same pixel at the same time, the counter is incremented by three. However, it is not possible to tell how many hits came from each waveform. To separate waveforms, you can position the waveforms vertically with the channel offset. By separating the waveforms, you can avoid overlapping data in the database caused by multiple waveforms. Even if the display is set to show only the most recent acquisition, the database keeps track of all pixel hits while the database is building.

Remember that color grade persistence, mask testing, and histograms all use the same database. Suppose that the database is building because color grade persistence is ON; when mask testing or histograms are turned on, they can use the information already established in the database as though they had been turned on the entire time.

To avoid erroneous data, clear the display after you change oscilloscope setup conditions or DUT conditions and acquire new data before extracting measurement results.

# AXIS

Command

```
:HISTogram:AXIS {VERTical | HORizontal}
```

The :HISTogram:AXIS command selects the type of histogram. A horizontal histogram can be used to measure time related information like jitter. A vertical histogram can be used to measure voltage related information like noise.

**Example**

This example defines a vertical histogram.

```
10 OUTPUT 707;":HISTOGRAM:AXIS VERTICAL"
20 END
```

Query

```
:HISTogram:AXIS?
```

The :HISTogram:AXIS? query returns the currently selected histogram type.

Returned Format

```
[:HISTogram:AXIS] {VERTical | HORizontal}<NL>
```

**Example**

This example returns the histogram type and prints it to the computer's screen.

```
10 DIM Axis$[50]
20 OUTPUT 707;":HISTOGRAM:AXIS?"
30 ENTER 707;Axis$
40 PRINT Axis$
50 END
```

# MODE

Command

```
:HISTogram:MODE {OFF | WAVeforms}
```

The :HISTogram:MODE command selects the histogram mode. The histogram may be off or set to track the waveform database.

**Example**

This example sets the histogram mode to track the waveform database.

```
10 OUTPUT 707;":HISTOGRAM:MODE WAVEFORM"
20 END
```

Query

```
:HISTogram:MODE?
```

The :HISTogram:MODE? query returns the currently selected histogram mode.

Returned Format

```
[:HISTogram:MODE] {OFF | WAVeform}<NL>
```

**Example**

This example returns the result of the mode query and prints it to the computer's screen.

```
10 DIM Mode$[10]
20 OUTPUT 707;":HISTOGRAM:MODE?"
30 ENTER 707;Mode$
40 PRINT Mode$
50 END
```

## SCALe:SIZE

Command

```
:HISTogram:SCALe:SIZE <size>
```

The :HISTogram:SCALe:SIZE command sets histogram size for vertical and horizontal mode.

`<size>` The size is from 1.0 to 8.0 for the horizontal mode and from 1.0 to 10.0 for the vertical mode.

**Example**

This example sets the histogram size to 3.5.

```
10 OUTPUT 707;":HISTOGRAM:SCALE:SIZE 3.5"
20 END
```

Query

```
:HISTogram:SCALe:SIZE?
```

The :HISTogram:SCALe:SIZE? query returns the correct size of the histogram.

Returned Format

```
[:HISTogram:SCALe:SIZE] <size><NL>
```

**Example**

This example returns the result of the size query and prints it to the computer's screen.

```
10 DIM Size$[50]
20 OUTPUT 707;":HISTOGRAM:SCALE:SIZE?"
30 ENTER 707;Size$
40 PRINT Size$
50 END
```

## WINDow:DEFault

Command

```
:HISTogram:WINDow:DEFault
```

The :HISTogram:WINDow:DEFault command positions the histogram markers to a default location on the display. Each marker will be positioned one division off the left, right, top, and bottom of the display.

**Example**

This example sets the histogram window to the default position.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:DEFAULT"
20 END
```

## WINDow:SOURce

Command

```
:HISTogram:WINDow:SOURce {CHANnel<N> | FUNCtion<N>
| WMEMory<N>}
```

The :HISTogram:WINDow:SOURce command selects the source of the histogram window. The histogram window will track the source's vertical and horizontal scale.

<N> For channels: the number represents an integer, 1 through 4.
For waveform memories: 1, 2, 3, or 4.
For functions: 1 or 2

**Example**

This example sets the histogram window's source to Channel 1.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:SOURCE CHANNEL1"
20 END
```

Query

```
:HISTogram:WINDow:SOURce?
```

The :HISTogram:WINDow:SOURce? query returns the currently selected histogram window source.

Returned Format

```
[:HISTogram:WINDow:SOURce] {CHANnelN | FUNCtionN |
WMEMoryN}<NL>
```

**Example**

This example returns the result of the window source query and prints it to the computer's screen.

```
10 DIM Winsour$[50]
20 OUTPUT 707;":HISTOGRAM:WINDOW:SOURCE?"
30 ENTER 707;Winsour$
40 PRINT Winsour$
50 END
```

## WINDow:X1Position | LLIMit

Command

```
:HISTogram:WINDow:X1Position <x1_position>

:HISTogram:WINDow:LLIMit <x1_position>
```

The :HISTogram:WINDow:X1Position command moves the X1 marker of the histogram window. The histogram window determines the portion of the display used to build the database for the histogram. The histogram window markers will track the scale of the histogram window source.

`<x1_position>` A real number that represents the left boundary of the histogram window.

**Example**

This example sets the X1 position to -200 microseconds.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:X1POSITION -200E-6"
20 END
```

Query

```
:HISTogram:WINDow:X1Position?

:HISTogram:WINDow:LLIMit?
```

The :HISTogram:WINDow:X1Position? query returns the value of the X1 histogram window marker.

Returned Format

```
[:HISTogram:WINDow:X1Position] <x1_position><NL>
```

**Example**

This example returns the result of the X1 position query and prints it to the computer's screen.

```
10 DIM X1$[50]
20 OUTPUT 707;":HISTOGRAM:WINDOW:X1POSITION?"
30 ENTER 707;X1$
40 PRINT X1$
50 END
```

## WINDow:X2Position | RLIMit

Command

```
:HISTogram:WINDow:X2Position <x2_position>

:HISTogram:WINDow:RLIMit <x2_position>
```

The :HISTogram:WINDow:X2Position command moves the X2 marker of the histogram window. The histogram window determines the portion of the display used to build the database used for the histogram. The histogram window markers will track the scale of the histogram window source.

`<x2_position>` A real number that represents the right boundary of the histogram window.

**Example**

This example sets the X2 marker to 200 microseconds.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:X2POSITION 200E-6"
20 END
```

Query

```
:HISTogram:WINDow:X2Position?

:HISTogram:WINDow:RLIMit?
```

The :HISTogram:WINDow:X2Position? query returns the value of the X2 histogram window marker.

Returned Format

```
[:HISTogram:WINDow:X2Position] <x2_position><NL>
```

**Example**

This example returns the result of the X2 position query and prints it to the computer's screen.

```
10 DIM X2$[50]
20 OUTPUT 707;":HISTOGRAM:WINDOW:X2POSITION?"
30 ENTER 707;X2$
40 PRINT X2$
50 END
```

## WINDow:Y1Position | BLIMit

Command

```
:HISTogram:WINDow:Y1Position <y1_POSITION>

:HISTogram:WINDow:BLIMit <y1_POSITION>
```

The :HISTogram:WINDow:Y1Position command moves the Y1 marker of the histogram window. The histogram window determines the portion of the display used to build the database used for the histogram. The histogram window markers will track the scale of the histogram window source.

`<y1_position>` A real number that represents the bottom boundary of the histogram window.

**Example**

This example sets the position of the Y1 marker to -250 mV.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:Y1POSITION -250E-3"
20 END
```

Query

```
:HISTogram:WINDow:Y1Position?

:HISTogram:WINDow:BLIMit?
```

The :HISTogram:WINDow:Y1Position? query returns the value of the Y1 histogram window marker.

Returned Format

```
[:HISTogram:WINDow:Y1Position] <y1_position><NL>
```

**Example**

This example returns the result of the Y1 position query and prints it to the computer's screen.

```
10 DIM Y1$[50]
20 OUTPUT 707;":HISTOGRAM:WINDOW:Y1POSITION?"
30 ENTER 707;Y1$
40 PRINT Y1$
50 END
```

## WINDow:Y2Position | TLIMit

Command

```
:HISTogram:WINDow:Y2Position <y2_position>

:HISTogram:WINDow:TLIMit <y2_position>
```

The :HISTogram:WINDow:Y2Position command moves the Y2 marker of the histogram window. The histogram window determines the portion of the display used to build the database used for the histogram. The histogram window markers will track the scale of the histogram window source.

<y2_position> A real number that represents the top boundary of the histogram window.

**Example**

This example sets the position of the Y2 marker to 250 mV.

```
10 OUTPUT 707;":HISTOGRAM:WINDOW:Y2POSITION 250E-3"
20 END
```

Query

```
:HISTogram:WINDow:Y2Position?

:HISTogram:WINDow:TLIMit?
```

The :HISTogram:WINDow:Y2Position? query returns the value of the Y2 histogram window marker.

Returned Format

```
[:HISTogram:WINDow:Y2Position] <y2_position><NL>
```

**Example**

This example returns the result of the Y2 position query and prints it to the computer's screen.

```
10 DIM Y2$[50]
20 OUTPUT 707;":HISTOGRAM:WINDOW:Y2POSITION?"
30 ENTER 707;Y2$
40 PRINT Y2$
50 END
```

19

# Marker Commands

# Marker Commands

The commands in the MARKer subsystem specify and query the settings of the time markers (X axis) and current measurement unit markers (volts, amps, and watts for the Y axis). You typically set the Y-axis measurement units using the :CHANnel:UNITs command.

These MARKer commands and queries are implemented in the Infiniium Oscilloscopes:

- CURsor?
- MEASurement:READout
- MODE
- TDELta?
- TSTArt
- TSTOp
- VDELta?
- VSTArt
- VSTOp
- X1Position
- X2Position
- X1Y1source
- X2Y2source
- XDELta?
- Y1Position
- Y2Position
- YDELta?

**Guidelines for Using Queries in Marker Modes**

In Track Waveforms mode, use :MARKer:CURSor? to track the position of the waveform. In Manual Markers and Track Measurements Markers modes, use other queries, such as the TSTArt? and TSTOp?, and VSTArt? and VSTOp? queries. If you use :MARKer:CURSor? when the oscilloscope is in either Manual Markers or Track Measurements Markers modes, it will put the oscilloscope in Track Waveforms mode, regardless of the mode previously selected.

# CURSor?

Query

```
:MARKer:CURSor? {DELTa | STARt | STOP}
```

The :MARKer:CURSor? query returns the time and current measurement unit values of the specified marker (if markers are in Track Waveforms mode) as an ordered pair of time and measurement unit values.

- If DELTA is specified, the value of delta Y and delta X are returned.
- If START is specified, marker A's x-to-y positions are returned.
- If STOP is specified, marker B's x-to-y positions are returned.

Returned Format

```
[:MARKer:CURSor] {DELTa | STARt | STOP}
{<Ax, Ay> | <Bx, By> | <deltaX, deltaY>}<NL>
```

**Example**

This example returns the current position of the X cursor and measurement unit marker 1 to the string variable, Position$. The program then prints the contents of the variable to the computer's screen.

```
10  DIM Position$[50]!Dimension variable
20  OUTPUT 707;":MARKER:CURSOR? START"
30  ENTER 707;Position$
40  PRINT Position$
50  END
```

**CAUTION**

**The :MARKer:CURSor? query may change marker mode and results.** In Track Waveforms mode, use :MARKer:CURSor? to track the position of the waveform. In Manual Markers and Track Measurements Markers modes, use other marker queries, such as the TSTArt? and TSTOp?, and VSTArt? and VSTOp? queries.

If you use :MARKer:CURSor? when the oscilloscope is in either Manual Markers or Track Measurements Markers modes, it will put the oscilloscope in Track Waveforms mode, regardless of the mode previously selected. In addition, measurement results may not be what you expected.

## MEASurement:READout

**Command**

```
:MARKer:MEASurement:READout {{ON|1} | {OFF|0}}
```

The :MARKer:MEASurement:READout command controls the display of the marker position values.

`ON|1` Shows marker position values.

`OFF|0` Turns off marker position values.

**Query**

```
:MARKer:MEASurement:READout?
```

The :MARKer:MEASurement:READout? query returns the current display of the marker position values.

**Returned Format**

```
{:MARKer:MEASurement:READout] {1|0}<NL>
```

**Example**

This example displays the marker position values.

```
10  OUTPUT 707;":MARKER:MEASUREMENT:READOUT ON"
20  END
```

# MODE

Command

```
:MARKer:MODE {OFF | MANual | WAVeform | MEASurement}
```

The :MARKer:MODE command sets the marker mode.

`OFF` Removes the marker information from the display.

`MANual` Enables manual placement of markers A and B.

`WAVeform` Tracks the current waveform.

`MEASurement` Tracks the most recent measurement.

**Example**

This example sets the marker mode to waveform.

```
10  OUTPUT 707;":MARKER:MODE WAVEFORM"
20  END
```

Query

```
:MARKer:MODE?
```

The :MARKer:MODE? query returns the current marker mode.

Returned Format

```
[:MARKer:MODE] {OFF | MANual | WAVeform | MEASurement}<NL>
```

**Example**

This example places the current marker mode in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10  DIM Selection$[50]!Dimension variable
20  OUTPUT 707;":MARKER:MODE?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

# TDELta?

Query

```
:MARKer:TDELta?
```

The :MARKer:TDELta? query returns the time difference between Ax and Bx time markers. The :MARKer:XDELta command described in this chapter does also.

**Use :MARKer:XDELta? Instead of :MARKer:TDELta?**

The :MARKer:TDELta? query performs the same function as the :MARKer:XDELta? query. The :MARKer:TDELta? query is provided for compatibility with programs written for older oscilloscopes. You should use :MARKer:XDELta? for new programs.

Returned Format

```
[:MARKer:TDELta] <time><NL>
```

<time> The time difference between Ax and Bx time markers.

**Example**

This example places the time difference between the Ax and Bx markers in the numeric variable, Time, then prints the contents of the variable to the computer's screen. Notice that this example uses the :MARKer:XDELta? query instead of the :MARKer:TDELta? query.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:XDELTA?"
30  ENTER 707;Time
40  PRINT Time
50  END
```

**Turn Headers Off**

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

# TSTArt

Command

```
:MARKer:TSTArt <Ax_position>
```

The :MARKer:TSTArt command sets the Ax marker position. The :MARKer:X1Position command described in this chapter also sets the Ax marker position.

**Use :MARKer:X1Position Instead of :MARKer:TSTArt**

The :MARKer:TSTArt command and query perform the same function as the :MARKer:X1Position command and query. The :MARKer:TSTArt command is provided for compatibility with programs written for previous oscilloscopes. You should use :MARKer:X1Position for new programs.

`<Ax_position>` A real number for the time at the Ax marker, in seconds.

**Example**

This example sets the Ax marker at 90 ns. Notice that this example uses the X1Position command instead of TSTArt.

```
10  OUTPUT 707;":MARKER:X1POSITION 90E-9"
20  END
```

Query

```
:MARKer:TSTArt?
```

The :MARKer:TSTArt? query returns the time at the Ax marker.

Returned Format

```
[:MARKer:TSTArt] <Ax_position><NL>
```

**Example**

This example places the current setting of the Ax marker in the numeric variable, Setting, then prints the contents of the variable to the computer's screen. Notice that this example uses the :MARKer:X1Position? query instead of the :MARKer:TSTArt? query.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off"
20  OUTPUT 707;":MARKER:X1POSITION?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

**Do Not Use TST as the Short Form of TSTArt and TSTOp**

The short form of the TSTArt command and query does not follow the defined convention for short form commands. Because the short form, TST, is the same for TSTArt and TSTOp, sending TST produces an error. Use TSTA for TSTArt.

# TSTOp

Command

```
:MARKer:TSTOp <Bx_position>
```

The :MARKer:TSTOp command sets the Bx marker position. The :MARKer:X2Position command described in this chapter also sets the Bx marker position.

**Use :MARKer:X2Position Instead of :MARKer:TSTOp**

The :MARKer:TSTOp command and query perform the same function as the :MARKer:X2Position command and query. The :MARKer:TSTOp command is provided for compatibility with programs written for previous oscilloscopes. You should use :MARKer:X2Position for new programs.

`<Bx_position>` A real number for the time at the Bx marker, in seconds.

**Example**

This example sets the Bx marker at 190 ns. Notice that this example uses the X2Position command instead of TSTOp.

```
10  OUTPUT 707;":MARKER:X2POSITION 190E-9"
20  END
```

Query

```
:MARKer:TSTOp?
```

The :MARKer:TSTOp? query returns the time at the Bx marker position.

Returned Format

```
[:MARKer:TSTOp] <Bx_position><NL>
```

**Example**

This example places the current setting of the Bx marker in the numeric variable, Setting, then prints the contents of the variable to the computer's screen. Notice that this example uses the :MARKer:X2Position? query instead of the :MARKer:TSTOp? query.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:X2POSITION?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

**Do Not Use TST as the Short Form of TSTArt and TSTOp**

The short form of the TSTOp command and query does not follow the defined convention for short form commands. Because the short form, TST, is the same for TSTArt and TSTOp, sending TST produces an error. Use TSTO for TSTOp.

# VDELta?

Query

```
:MARKer:VDELta?
```

The :MARKer:VDELta? query returns the current measurement unit difference between markers Ay and By. The :MARKer:YDELta? query described in this chapter does also.

**Use :MARKer:YDELta? Instead of :MARKer:VDELta?**

The :MARKer:VDELta? query performs the same function as the :MARKer:YDELta? query. The :MARKer:VDELta? query is provided for compatibility with programs written for previous oscilloscopes. You should use the :MARKer:YDELta? query for new programs.

Returned Format

```
[:MARKer:VDELta] <value><NL>
```

`<value>` Current measurement unit difference between markers Ay and By.

**Example**

This example returns the voltage difference between Ay and By to the numeric variable, Volts, then prints the contents of the variable to the computer's screen. Notice that this example uses the :MARKer:YDELta? query instead of the :MARKer:VDELta? query.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:YDELTA?"
30  ENTER 707;Volts
40  PRINT Volts
50  END
```

# VSTArt

Command

```
:MARKer:VSTArt <Ay_position>
```

The :MARKer:VSTArt command sets the Ay marker position and moves the Ay marker to the specified measurement unit value on the specified source. The :MARKer:Y1Position command described in this chapter does also.

**Use :MARKer:Y1Position Instead of :MARKer:VSTArt**

The :MARKer:VSTArt command and query perform the same function as the :MARKer:Y1Position command and query. The :MARKer:VSTArt command is provided for compatibility with programs written for previous oscilloscopes. You should use :MARKer:Y1Position for new programs.

`<Ay_position>` A real number for the current measurement unit value at Ay (volts, amps, or watts).

**Example**

This example sets Ay to –10 mV. Notice that this example uses the Y1Position command instead of VSTArt.

```
10  OUTPUT 707;":MARKER:Y1POSITION -10E-3"
20  END
```

Query

```
:MARKer:VSTArt?
```

The :MARKer:VSTArt? query returns the current measurement unit level of Ay.

Returned Format

```
[:MARKer:VSTArt] <Ay_position><NL>
```

**Example**

This example returns the voltage setting for Ay to the numeric variable, Value, then prints the contents of the variable to the computer's screen. Notice that this example uses the :MARKer:Y1Position? query instead of the :MARKer:VSTArt? query.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:Y1POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

**Do Not Use VST as the Short Form of VSTArt and VSTOp**

The short form of the VSTArt command and query does not follow the defined convention for short form commands. Because the short form, VST, is the same for VSTArt and VSTOp, sending VST produces an error. Use VSTA for VSTArt.

# VSTOp

Command

```
:MARKer:VSTOp <By_position>
```

The :MARKer:VSTOp command sets the By marker position and moves By to the specified measurement unit on the specified source.

The :MARKer:Y2Position command described in this chapter does also.

**Use :MARKer:Y2Position Instead of :MARKer:VSTOp**

The :MARKer:VSTOp command and query perform the same function as the :MARKer:Y2Position command and query. The :MARKer:VSTOp command is provided for compatibility with programs written for previous oscilloscopes. You should use :MARKer:Y2Position for new programs.

`<By_position>` A real number for the current measurement unit value at By (volts, amps, or watts).

**Example**

This example sets By to -100 mV. Notice that this example uses the :MARKer:Y2Position command instead of :MARKer:VSTOp.

```
10  OUTPUT 707;":MARKER:Y2POSITION -100E-3"
20  END
```

Query

```
:MARKer:VSTOp?
```

The :MARKer:VSTOp? query returns the current measurement unit level at By.

Returned Format

```
[:MARKer:VSTOp] <By_position><NL>
```

**Example**

This example returns the voltage at By to the numeric variable, Value, then prints the contents of the variable to the computer's screen. Notice that this example uses the :MARKer:Y2Position? query instead of the :MARKer:VSTOp? query.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:Y2POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

**Do Not Use VST as the Short Form of VSTArt and VSTOp**

The short form of the VSTOp command and query does not follow the defined convention for short form commands. Because the short form, VST, is the same for VSTArt and VSTOp, sending VST produces an error. Use VSTO for VSTOp.

# X1Position

Command

```
:MARKer:X1Position <Ax_position>
```

The :MARKer:X1Position command sets the Ax marker position, and moves the Ax marker to the specified time with respect to the trigger time.

`<Ax_position>` A real number for the time at the Ax marker in seconds.

**Example**

This example sets the Ax marker to 90 ns.

```
10  OUTPUT 707;":MARKER:X1POSITION 90E-9"
20  END
```

Query

```
:MARKer:X1Position?
```

The :MARKer:X1Position? query returns the time at the Ax marker position.

Returned Format

```
[:MARKer:X1Position] <Ax_position><NL>
```

**Example**

This example returns the current setting of the Ax marker to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:X1POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

**See Also**

:MARKer:TSTArt

# X2Position

Command

```
:MARKer:X2Position <Bx_position>
```

The :MARKer:X2Position command sets the Bx marker position and moves the Bx marker to the specified time with respect to the trigger time.

`<Bx_position>` A real number for the time at the Bx marker in seconds.

**Example**

This example sets the Bx marker to 90 ns.

```
10  OUTPUT 707;":MARKER:X2POSITION 90E-9"
20  END
```

Query

```
:MARKer:X2Position?
```

The :MARKer:X2Position? query returns the time at Bx marker in seconds.

Returned Format

```
[:MARKer:X2Position] <Bx_position><NL>
```

**Example**

This example returns the current position of the Bx marker to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:X2POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# X1Y1source

Command

```
:MARKer:X1Y1source {CHANnel<N> | FUNCtion<N> |
WMEMory<N>}
```

The :MARKer:X1Y1source command sets the source for the Ax and Ay markers. The channel you specify must be enabled for markers to be displayed. If the channel, function, or waveform memory that you specify is not on, an error message is issued and the query will return channel 1.

<N> CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

Integers, 1 - 4, representing the selected function or waveform memory.

**Example**

This example selects channel 1 as the source for markers Ax and Ay.

```
10  OUTPUT 707;":MARKER:X1Y1SOURCE CHANNEL1"
20  END
```

Query

```
:MARKer:X1Y1source?
```

The :MARKer:X1Y1source? query returns the current source for markers Ax and Ay.

Returned Format

```
[:MARKer:X1Y1source] {CHANnel<N> | FUNCtion<N> |
WMEMory<N>}<NL>
```

**Example**

This example returns the current source selection for the Ax and Ay markers to the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10  DIM Selection$[50]!Dimension variable
20  OUTPUT 707;":MARKER:X1Y1SOURCE?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

# X2Y2source

Command

```
:MARKer:X2Y2source {CHANnel<N> | FUNCtion<N> |
WMEMory<N>}
```

The :MARKer:X2Y2source command sets the source for the Bx and By markers. The channel you specify must be enabled for markers to be displayed. If the channel, function, or waveform memory that you specify is not on, an error message is issued and the query will return channel 1.

<N> CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

Integers, 1 - 4, representing the selected function or waveform memory.

**Example**

This example selects channel 1 as the source for markers Bx and By.

```
10  OUTPUT 707;":MARKER:X2Y2SOURCE CHANNEL1"
20  END
```

Query

```
:MARKer:X2Y2source?
```

The :MARKer:X2Y2source? query returns the current source for markers Bx and By.

Returned Format

```
[:MARKer:X2Y2source] {CHANnel<N> | FUNCtion<N> |
WMEMory<N>}<NL>
```

**Example**

This example returns the current source selection for the Bx and By markers to the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10  DIM Selection$[50]!Dimension variable
20  OUTPUT 707;":MARKER:X2Y2SOURCE?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

# XDELta?

Query

```
:MARKer:XDELta?
```

The :MARKer:XDELta? query returns the time difference between Ax and Bx time markers.

Xdelta = time at Bx – time at Ax

Returned Format

```
[:MARKer:XDELta] <time><NL>
```

<time> Time difference between Ax and Bx time markers in seconds.

**Example**

This example returns the current time between the Ax and Bx time markers to the numeric variable, Time, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:XDELTA?"
30  ENTER 707;Time
40  PRINT Time
50  END
```

# Y1Position

Command

```
:MARKer:Y1Position <Ay_position>
```

The :MARKer:Y1Position command sets the Ay marker position on the specified source.

`<Ay_position>` A real number for the current measurement unit value at Ay (volts, amps, or watts).

**Example**

This example sets the Ay marker to 10 mV.

```
10  OUTPUT 707;":MARKER:Y1POSITION 10E-3"
20  END
```

Query

```
:MARKer:Y1Position?
```

The :MARKer:Y1Position? query returns the current measurement unit level at the Ay marker position.

Returned Format

```
[:MARKer:Y1Position] <Ay_position><NL>
```

**Example**

This example returns the current setting of the Ay marker to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:Y1POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# Y2Position

Command

```
:MARKer:Y2Position <By_position>
```

The :MARKer:Y2Position command sets the By marker position on the specified source.

<By_position> A real number for the current measurement unit value at By (volts, amps, or watts).

**Example**

This example sets the By marker to -100 mV.

```
10  OUTPUT 707;":MARKER:Y2POSITION -100E-3"
20  END
```

Query

```
:MARKer:Y2Position?
```

The :MARKer:Y2Position? query returns the current measurement unit level at the By marker position.

Returned Format

```
[:MARKer:Y2Position] <By_position><NL>
```

**Example**

This example returns the current setting of the By marker to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:Y2POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

## YDELta?

Query

```
:MARKer:YDELta?
```

The :MARKer:YDELta? query returns the current measurement unit difference between Ay and By.

Vdelta = value at By – value at Ay

Returned Format

```
[:MARKer:YDELta] <value><NL>
```

`<value>` Measurement unit difference between Ay and By.

**Example**

This example returns the voltage difference between Ay and By to the numeric variable, Volts, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MARKER:YDELTA?"
30  ENTER 707;Volts
40  PRINT Volts
50  END
```

20

# Measure Commands

# Measure Commands

The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms.

These MEASure commands and queries are implemented in the Infiniium Oscilloscopes:

- AREA
- CGRade:CROSsing
- CGRade:DCDistortion
- CGRade:EHEight
- CGRade:EWIDth
- CGRade:JITTer
- CGRade:QFACtor
- CLEar | SCRatch
- CTCJitter (available only on the 54845A and 54846A)
- DEFine
- DELTatime
- DUTYcycle
- FALLtime
- FFT:DFRequency (delta frequency)
- FFT:DMAGnitude (delta magnitude)
- FFT:FREQuency
- FFT:MAGNitude
- FFT:PEAK1
- FFT:PEAK2
- FFT:THReshold
- FREQuency
- HISTogram:HITS
- HISTogram:MEAN
- HISTogram:MEDian
- HISTogram:M1S
- HISTogram:M2S

- HISTogram:M3S
- HISTogram:PEAK
- HISTogram:PP
- HISTogram:STDDev
- JITTer (available only on the 54845A and 54846A)
- NWIDth
- OVERshoot
- PERiod
- PHAse
- PREShoot
- PWIDth
- RESults?
- RISetime
- SCRatch | CLEar
- SENDvalid
- SOURce
- STATistics
- TEDGe
- TMAX
- TMIN
- TVOLt
- VAMPlitude
- VAVerage
- VBASe
- VLOWer
- VMAX
- VMIDdle
- VMIN
- VPP
- VRMS
- VTIMe
- VTOP
- VUPPer

### FFT Commands

The :MEASure:FFT commands control the FFT measurements that are accessible through the Measure subsystem.

### Measurement Setup

To make a measurement, the portion of the waveform required for that measurement must be displayed on the oscilloscope.

- For a period or frequency measurement, at least one and a half complete cycles must be displayed.
- For a pulse width measurement, the entire pulse must be displayed.
- For a rise time measurement, the leading (positive-going) edge of the waveform must be displayed.
- For a fall time measurement, the trailing (negative-going) edge of the waveform must be displayed.

### User-Defined Measurements

When you make user-defined measurements, the defined parameters must be set before actually sending the measurement command or query.

### Measurement Error

If a measurement cannot be made because of a lack of data, because the source waveform is not displayed, the requested measurement is not possible (for example, a period measurement on an FFT waveform), or for some other reason, the following results are returned:

- 9.99999E+37 is returned as the measurement result.
- If SENDvalid is ON, the error code is also returned.

### Making Measurements

If more than one period, edge, or pulse is displayed, time measurements are made on the first, left-most portion of the displayed waveform.

When any of the defined measurements are requested, the oscilloscope first determines the top (100%) and base (0%) voltages of the waveform. From this information, the oscilloscope determines the other important voltage values (10%, 90%, and 50% voltage values) for making measurements.

The 10% and 90% voltage values are used in the rise time and fall time measurements when standard measurements are selected. The 50% voltage value is used for measuring frequency, period, pulse width, and duty cycle with standard measurements selected.

You can also make measurements using user-defined parameters instead of the standard measurement values.

When the command form of a measurement is used, the oscilloscope is placed in the continuous measurement mode. The measurement result will be displayed on the front panel. There may be a maximum of four measurements running continuously. Use the SCRatch command to turn off the measurements.

When the query form of the measurement is used, the measurement is made one time, and the measurement result is returned.

- If the current acquisition is complete, the current acquisition is measured and the result is returned.
- If the current acquisition is incomplete and the oscilloscope is running, acquisitions will continue to occur until the acquisition is complete. The acquisition will then be measured and the result returned.
- If the current acquisition is incomplete and the oscilloscope is stopped, the measurement result will be 9.99999e+37 and the incomplete result state will be returned if SENDvalid is ON.

All measurements are made using the entire display, except for VAVerage and VRMS which allow measurements on a single cycle. Therefore, if you want to make measurements on a particular cycle, display only that cycle on the screen.

Measurements are made on the displayed waveforms specified by the SOURce command. The SOURce command lets you specify two sources. Most measurements are only made on a single source. Some measurements, such as the DELTatime measurement, require two sources.

If the waveform is clipped, the measurement result may be questionable. In this case, the value returned is the most accurate value that can be made using the current scaling. You might be able to obtain a more accurate measurement by adjusting the vertical scale to prevent the waveform from being clipped.

# AREA

Command

```
:MEASure:AREA {CYCle | DISPlay}[,<source>]
```

The :MEASure:AREA command turns on the area measurement. The area measurement measures between the waveform, or a selected cycle of the waveform, and the waveform ground. When measuring Area, it is sometimes useful to use the Subtract Math Operator to remove any dc offset from a waveform you want to measure. Also see Math/FFT Functions for more details.

<source> {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

<N> CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example turns on the area measurement which measures between the waveform and ground. Only that portion of the waveform which is in the waveform viewing area is measured.

```
10 OUTPUT 707;"MEASURE:AREA DISPLAY"
20 END
```

Query

```
:MEASure:AREA?
```

The :MEASure:AREA? query returns the area measurement.

Returned Format

```
[:MEASure:AREA]<value>[,<result_state>]<NL>
```

**Example**

This example places the current selection for the area to be measured in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10 DIM Selection$[50]
20 OUTPUT 707;"MEASure:AREA?"
30 ENTER 707;Selection$
40 PRINT Selection$
50 END
```

# CGRade:CROSsing

Command

```
:MEASure:CGRade:CROSsing
```

The :MEASure:CGRade:CROSsing command enables the crossing level percent measurement on the current eye pattern. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

**Example**

This example measures the crossing level.

```
10 OUTPUT 707;"MEASURE:CGRADE:CROSSING"
20 END
```

Query

```
:MEASure:CGRade:CROSsing?
```

The :MEASure:CGRade:CROSsing? query returns the crossing level percent measurement of the current eye diagram on the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:CROSsing]<value>[,<result_state>]<NL>
```

`<value>` The crossing level.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example places the current crossing level in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"        !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:CROSSING?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

# CGRade:DCDistortion

Command

```
:MEASure:CGRade:DCDistortion <format>
```

The :MEASure:CGRade:DCDistortion command enables the duty cycle distortion measurement on the current eye pattern. The parameter specifies the format for reporting the measurement. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

`<format>` {TIME | PERCent}

**Example**

This example measures the duty cycle distortion.

```
10 OUTPUT 707;"MEASURE:CGRADE:DCDISTORTION TIME"
20 END
```

Query

```
:MEASure:CGRade:DCDistortion? <format>
```

The :MEASure:CGRade:DCDistortion query returns the duty cycle distortion measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:DCDistortion]<value>[,<result_state>]<NL>
```

`<value>` The duty cycle distortion.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example places the current duty cycle distortion in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"
20 OUTPUT 707;":MEASURE:CGRADE:DCDISTORTION? PERCENT"
30 ENTER 707;Value
40 PRINT Value
50 END
```

# CGRade:EHEight

Command

```
:MEASure:CGRade:EHEight <format>
```

The :MEASure:CGRade:EHEight command enables the eye height measurement on the current eye pattern. The parameter specifies the format for reporting the measurement. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

`<format>` {TIME | PERCent}

**Example**

This example measures the eye height.

```
10 OUTPUT 707;"MEASURE:CGRADE:EHEIGHT TIME"
20 END
```

Query

```
:MEASure:CGRade:EHEight? <format>
```

The :MEASure:CGRade:EHEight? query returns the eye height measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:EHEight]<value>[,<result_state>]<NL>
```

`<value>` The eye height.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example places the current eye height in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:EHEIGHT?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

# CGRade:EWIDth

Command

```
:MEASure:CGRade:EWIDth
```

The :MEASure:CGRade:EWIDth command enables the eye width measurement on the current eye pattern. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

**Example**

This example measures the eye width.

```
10 OUTPUT 707;"MEASURE:CGRADE:EWIDTH"
20 END
```

Query

```
:MEASure:CGRade:EWIDth?
```

The :MEASure:CGRade:EWIDth? query returns the eye width measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:EWIDth]<value>[,<result_state>]<NL>
```

`<value>` The eye width.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example places the current eye width in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:EWIDTH?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

## CGRade:JITTer

Command

```
:MEASure:CGRade:JITTer <format>
```

The :MEASure:CGRade:JITTer measures the jitter at the eye diagram crossing point. The parameter specifies the format, peak-to-peak or RMS, of the returned results. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

`<format>` {PP | RMS}

**Example**

This example measures the jitter.

```
10 OUTPUT 707;"MEASURE:CGRADE:JITTER RMS"
20 END
```

Query

```
:MEASure:CGRade:JITTer? <format>
```

The :MEASure:CGRade:JITTer? query returns the jitter measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:JITTer]<value>[,<result_state>]<NL>
```

`<value>` The jitter.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example places the current jitter in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:JITTER? RMS"
30 ENTER 707;Value
40 PRINT Value
50 END
```

# CGRade:QFACtor

Command

```
:MEASure:CGRade:QFACtor
```

The :MEASure:CGRade:QFACtor command measures the Q factor. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

**Example**

This example measures the Q factor.

```
10 OUTPUT 707;"MEASURE:CGRADe:QFACTOR"
20 END
```

Query

```
:MEASure:CGRade:QFACtor?
```

The :MEASure:CGRade:QFACtor? query returns the Q factor measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:QFACtor]<value>[,<result_state>]<NL>
```

`<value>` The Q factor.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example places the Q factor in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"       !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:QFACTOR"
30 ENTER 707;Value
40 PRINT Value
50 END
```

# CLEar

Command

```
:MEASure:{CLEar | SCRatch}
```

The :MEASure:CLEar command clears the measurement results from the screen and disables all previously enabled measurements.

**Example**

This example clears the current measurement results from the screen.

```
10 OUTPUT 707;":MEASURE:CLEAR"
20 END
```

# CTCJitter

Command

```
:MEASure:CTCJitter [<source>]
```

The :MEASure:CTCJitter command measures the cycle-to-cycle jitter for the specified channel, waveform memory, or function. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:CTCJitter command.

This command is only available on the 54845A and 54846A oscilloscopes.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is: integer, 1 - 4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the cycle-to-cycle jitter of channel 1.

```
10 OUTPUT 707;"MEASURE:CTCJITTER CHANNEL1"
20 END
```

Query

```
:MEASure:CTCJitter? [<source>]
```

The :MEASure:CTCJitter? query returns the cycle-to-cycle jitter measurement for the selected source. Before using this query, you must use the :MEASure:CTCJitter command to enable the cycle-to-cycle jitter measurement feature.

This query is only available on the 54845A and 54846A oscilloscopes.

Returned Format

```
[:MEASure:CTCJitter] <value>[,<result_state>]<NL>
```

<value> The cycle-to-cycle jitter measurement.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example places the cycle-to-cycle jitter measurement in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:CGRADE:QFACTOR"
30 ENTER 707;Value
40 PRINT Value
50 END
```

# DEFine

Command

```
:MEASure:DEFine <meas_spec>
```

The :MEASure:DEFine command sets up the definition for measurements by specifying the delta time, threshold, or top-base values. Changing these values may affect other measure commands. Table 20-1 identifies the relationships between user-defined values and other MEASure commands.

`<meas_spec>` `{DELTatime | EWINdow | THResholds | TOPBase }`

**Table 20-1** **:MEASure:DEFine Interactions**

| MEASure Commands | DELTatime | THResholds | TOPBase |
|---|---|---|---|
| RISEtime | | x | x |
| FALLtime | | x | x |
| PERiod | | x | x |
| FREQuency | | x | x |
| VTOP | | | x |
| VBASe | | | x |
| VAMPlitude | | | x |
| PWIDth | | x | x |
| NWIDth | | x | x |
| OVERshoot | | x | x |
| DUTYcycle | | x | x |
| DELTatime | x | x | x |
| VRMS | | x | x |
| PREShoot | | x | x |
| VLOWer | | x | x |
| VMIDdle | | x | x |
| VUPPer | | x | x |
| VAVerage | | x | x |
| VARea | | x | x |

Command `:MEASure:DEFine DELTatime,<start_edge_direction>, <start_edge_number>,<start_edge_position>, <stop_edge_direction>,<stop_edge_number>, <stop_edge_position>`

`<edge_direction>` {RISing | FALLing | EITHer} for start and stop directions.

`<edge_number>` An integer from 1 to 65534 for start and stop edge numbers.

`<edge_position>` {UPPer | MIDDle | LOWer} for start and stop edge positions.

Command `:MEASure:DEFine EWINdow,<start>,<stop> [,<start_after>]`

`<start>` An integer from 1 to 100 for horizontal starting point. (Default value is 40%.)

`<stop>` An integer from 1 to 100 for horizontal stopping point. (Default value is 60%.)

`<start_after>` An integer from 1 to 63,488 for number of hits to acquire before making measurements. (Default value is 1.)

Command `:MEASure:DEFine THResholds,{{STANdard}| {PERCent,<upper_pct>,<middle_pct>,<lower_pct>} | {VOLTage,<upper_volts>,<middle_volts>, <lower_volts>}},{ALL|CHANnel<N>| FUNCtion<N>|WMEMory<N>}`

`<upper_pct>`
`<middle_pct>`
`<lower_pct>` An integer, - 25 to 125.

`<upper_volts>`
`<middle_volts>`
`<lower_volts>` A real number specifying voltage.

Command

```
:MEASure:DEFine TOPBase,{{STANdard}
|{<top_volts>,<base_volts>}},{ALL|CHANnel<N>|
FUNCtion<N>|WMEMory<N>}
```

`<top_volts>`
`<base_volts>` A real number specifying voltage.

**Example**

This example sets the parameters for a time measurement from the first positive edge at the upper threshold level to the second negative edge at the middle threshold.

```
10  OUTPUT 707;":MEASURE:DEFINE DELTATIME,RISING,
1,UPPER,FALLING,2,MIDDLE"
20  END
```

If you specify one source, both parameters apply to that waveform. If you specify two sources, the measurement is from the first positive edge on source 1 to the second negative edge on source 2.

Specify the source either using :MEASure:SOURce, or using the optional <source> parameter when the DELTatime measurement is started.

Query

```
:MEASure:DEFine? {DELTatime | EWINdow | THResholds|
TOPBase}<start>
```

The :MEASure:DEFine? query returns the current setup for the specified parameter.

Returned Format

```
[:MEASure:DEFine DELTatime] <start_edge_direction>,
<start_edge_number>,<start_edge_position>,
<stop_edge_direction>,<stop_edge_number>,
<stop_edge_position><NL>
```

```
[:MEASure:DEFine] EWINdow,<start>,<stop>,<start_after> <NL>
```

```
[:MEASure:DEFine] THResholds,{{STANdard} |
{PERcent,<upper_pct>,<middle_pct>,<lower_pct>} |
{VOLTage,<upper_volts>,<middle_volts>,<lower_volts>}}<NL>,
{ALL|CHANnel<N>|FUNCtion<N>|WMEMory<N>}
```

```
[:MEASure:DEFine] TOPBase,{{STANdard}
|{<top_volts>,<base_volts>}}<NL>,{ALL|CHANnel<N>|
FUNCtion<N>|WMEMory<N>}
```

**Use the Suffix Multiplier Instead**

Using "mV" or "V" following the numeric value for the voltage value will cause Error 138 - Suffix not allowed. Instead, use the convention for the suffix multiplier as described in chapter 3, "Message Communication and System Functions."

**Example**

This example returns the current setup for the measurement thresholds to the string variable, Setup$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setup$[50]!Dimension variable
20  OUTPUT 707;":MEASURE:DEFINE? THRESHOLDS"
30  ENTER 707; Setup$
40  PRINT Setup$
50  END
```

# DELTatime

Command

```
:MEASure:DELTatime [<source>[,<source>]]
```

The :MEASure:DELTatime command measures the delta time between two edges. If one source is specified, the delta time from the leading edge of the specified source to the trailing edge of the specified source is measured. If two sources are specified, the delta time from the leading edge on the first source to the trailing edge on the second source is measured.

Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:DELTatime command. The rest of the parameters for this command are specified with the :MEASure:DEFine command.

The necessary waveform edges must be present on the display. The query will return 9.99999E+37 if the necessary edges are not displayed.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the delta time between channel 1 and channel 2.

```
10  OUTPUT 707;":MEASURE:DELTATIME CHANNEL1,CHANNEL2"
20  END
```

Query `:MEASure:DELTatime? [<source>[,<source>]]`

The :MEASure:DELTatime? query returns the measured delta time value.

Returned Format `[:MEASure:DELTatime] <value>[,<result_state>]<NL>`

`<value>` Delta time from the first specified edge on one source to the next specified edge on another source.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current value of delta time in the numeric variable, Value, then prints the contents of the variable to the computer's screen. This example assumes the source was set using :MEASure:SOURce.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:DELTATIME?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

**Turn Off Headers**

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

**Related Commands** :MEASure:DEFine DELTatime

# DUTYcycle

Command

```
:MEASure:DUTYcycle [<source>]
```

The :MEASure:DUTYcycle command measures the ratio of the positive pulse width to the period. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:DUTYcycle command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the duty cycle of the last specified waveform.

```
10  OUTPUT 707;":MEASURE:DUTYCYCLE"
20  END
```

Query

```
:MEASure:DUTYcycle? [<source>]
```

The :MEASure:DUTYcycle? query returns the measured duty cycle of the specified source.

Returned Format

```
[:MEASure:DUTYcycle] <value>[,<result_state>]<NL>
```

`<value>` The ratio of the positive pulse width to the period.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current duty cycle of the specified waveform in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:DUTYCYCLE?
30  ENTER 707;Value
40  PRINT Value
50  END
```

# FALLtime

Command

```
:MEASure:FALLtime [<source>]
```

The :MEASure:FALLtime command measures the time at the upper threshold of the falling edge, measures the time at the lower threshold of the falling edge, then calculates the fall time. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FALLtime command.

The first displayed falling edge is used for the fall-time measurement. Therefore, for best measurement accuracy, set the sweep speed as fast as possible while leaving the falling edge of the waveform on the display.

Fall time = time at lower threshold point – time at upper threshold point.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the fall time of the last specified waveform.

```
10  OUTPUT 707;":MEASURE:FALLTIME"
20  END
```

Query `:MEASure:FALLtime?[<source>]`

The :MEASure:FALLtime? query returns the fall time of the specified source.

Returned Format `[:MEASure:FALLtime] <value>[,<result_state>]<NL>`

`<value>` Time at lower threshold - time at upper threshold.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current value for fall time in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:FALLTIME?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# FFT:DFRequency

**Command**

```
:MEASure:FFT:DFRequency [<source>]
```

The :MEASure:FFT:DFRequency command enables the delta frequency measurement. The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FFT command.

The source must be a function that is set to FFT, or a waveform memory that contains an FFT for this command and query to work.

`<source>` {FUNCtion<N> | WMEMory<N>}

`<N>` For functions and waveform memories: 1, 2, 3, or 4.

**Query**

```
:MEASure:FFT:DFRequency? [<source>]
```

The :MEASure:FFT:DFRequency? query returns the FFT delta frequency of the specified peaks.

**Returned Format**

```
[:MEASure:FFT:DFRequency]
<delta_frequency>[,<result_state>]<NL>
```

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Related Commands** :MEASure:FFT:PEAK1, :MEASure:FFT:PEAK2, :MEASure:FFT:THReshold

**Example**

This example measures the frequency difference between peaks 2 and 3 of an FFT of channel 4.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":func4:fftm chan4"!Perform FFT on channel 4
30  OUTPUT 707;":func4:disp on"!Display the FFT
40  OUTPUT 707;":meas:FFT:thr -47"!Set peak threshold at -47 dBm
50  OUTPUT 707;":meas:FFT:Peak1 2"!Meas diff between peak 2 and 3
60  OUTPUT 707;":meas:FFT:Peak2 3"
70  OUTPUT 707;":meas:FFT:dfr func4"!Perform dfrequency meas
80  OUTPUT 707;":meas:FFT:dfr? func4"!Query oscilloscope for
measurement
90  ENTER 707;Frequency
100 PRINT Frequency
110 END
```

# FFT:DMAGnitude

**Command**

```
:MEASure:FFT:DMAGnitude [<source>]
```

The :MEASure:FFT:DMAGnitude command enables the delta magnitude measurement. The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FFT command.

The source must be a function that is set to FFT, or a waveform memory that contains an FFT for this command and query to work.

`<source>` {FUNCtion<N> | WMEMory<N>}

`<N>` For functions and waveform memories: 1, 2, 3, or 4.

**Query**

```
:MEASure:FFT:DMAGnitude? [<source>]
```

The :MEASure:FFT:DMAGnitude? query returns the delta magnitude of the specified peaks.

**Returned Format**

```
[:MEASure:FFT:DMAGnitude]
<delta_magnitude>[,<result_state>]<NL>
```

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Related Commands** :MEASure:FFT:PEAK1, :MEASure:FFT:PEAK2, :MEASure:FFT:THReshold

## FFT:FREQuency

Command

```
:MEASure:FFT:FREQuency [<source>]
```

The :MEASure:FFT:FREQuency command enables the frequency measurement. The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FFT command.

The source must be a function that is set to FFT, or a waveform memory that contains an FFT for this command and query to work.

`<source>` {FUNCtion<N> | WMEMory<N>}

`<N>` For functions and waveform memories: 1, 2, 3, or 4.

Query

```
:MEASure:FFT:FREQuency? [<source>]
```

The :MEASure:FFT:FREQuency? query returns the frequency measurement.

Returned Format

```
[:MEASure:FFT:FREQuency] <frequency>[,<result_state>]<NL>
```

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

# FFT:MAGNitude

**Command**

```
:MEASure:FFT:MAGNitude [<source>]
```

The :MEASure:FFT:MAGNitude command measures the magnitude of the FFT. The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FFT command.

The source must be a function that is set to FFT, or a waveform memory that contains an FFT for this command and query to work.

`<source>` {FUNCtion<N> | WMEMory<N>}

`<N>` For functions and waveform memories: 1, 2, 3, or 4.

**Query**

```
:MEASure:FFT:MAGNitude?
```

The :MEASure:FFT:MAGNitude? query returns the magnitude value of the FFT.

**Returned Format**

```
[:MEASure:FFT:FMAGNitude] <magnitude>[,<result_state>]<NL>
```

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

# FFT:PEAK1

Command

```
:MEASure:FFT:PEAK1 <1st_peak_number>
```

The :MEASure:FFT:PEAK1command sets the peak number of the first peak for FFT measurements. The source is specified with the :MEASure:SOURce command as FUNCtion<N> or WMEMory<N>.

`<1st_peak_number>` An integer, 1 to 100,000 specifying the number of the first peak.

`<N>` For functions and waveform memories: 1, 2, 3, or 4.

Query

```
:MEASure:FFT:PEAK1?
```

The :MEASure:FFT:PEAK1? query returns the peak number currently set as the first peak.

Returned Format

```
[:MEASure:FFT:PEAK1] <1st_peak_number><NL>
```

**See Also**

:MEASure:FFT:THReshold

Also see the example for :MEASure:FFT:DFRequency in this chapter.

# FFT:PEAK2

**Command**

```
:MEASure:FFT:PEAK2 <2nd_peak_number>
```

The :MEASure:FFT:PEAK2 command sets the peak number of the second peak for FFT measurements. The source is specified with the :MEASure:SOURce command as FUNCtion<N> or WMEMory<N>.

`<2nd_peak_number>` An integer, 1 to 100,000 specifying the number of the second peak.

`<N>` For functions and waveform memories: 1, 2, 3, or 4.

**Query**

```
:MEASure:FFT:PEAK2?
```

The :MEASure:FFT:PEAK2? query returns the peak number currently set as the second peak.

**Returned Format**

```
[:MEASure:FFT:PEAK1] <2nd_peak_number><NL>
```

**See Also**

:MEASure:FFT:THReshold

Also see the example for :MEASure:FFT:DFRequency in this chapter.

## FFT:THReshold

**Command**

```
:MEASure:FFT:THReshold  <threshold_value>
```

The :MEASure:FFT:THReshold command sets the peak search threshold value in dB. The dB after the threshold value is optional.

`<threshold_value>` A real number specifying the threshold for peaks.

**Query**

```
:MEASure:FFT:THReshold?
```

The :MEASure:FFT:THReshold? query returns the peak search threshold value.

**Returned Format**

```
[:MEASure:FFT:THReshold] <threshold_value><NL>
```

These :MEASure commands also operate on FFT functions:

| Measure Command | Measurement Performed |
|---|---|
| :TMAX | The frequency of the maximum value in the spectrum. |
| :TMIN | The frequency of the minimum value in the spectrum. |
| :VMAX | The maximum value in the spectrum. |
| :VMIN | The minimum value in the spectrum. |
| :VPP | The range of values in the spectrum. |
| :VTIM | The value at a specified frequency. |

**See Also**

Also see the example for :MEASure:FFT:DFRequency in this chapter.

# FREQuency

Command

```
:MEASure:FREQuency [<source>]
```

The :MEASure:FREQuency command measures the frequency of the first complete cycle on the screen using the mid-threshold levels of the waveform (50% levels if standard measurements are selected). The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FREQuency command.

The algorithm is:

If the first edge on the screen is rising,

then

frequency = 1/(time at second rising edge - time at first rising edge)

else

frequency = 1/(time at second falling edge - time at first falling edge).

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the frequency of the last specified waveform.

```
10  OUTPUT 707;":MEASURE:FREQUENCY"
20  END
```

Query

```
:MEASure:FREQuency? [<source>]
```

The :MEASure:FREQuency? query returns the measured frequency.

Returned Format

```
[:MEASure:FREQuency] <value>[,<result_state>]<NL>
```

<value> The frequency value in Hertz of the first complete cycle on the screen using the mid-threshold levels of the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current frequency of the waveform in the numeric variable, Freq, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:FREQUENCY?"
30  ENTER 707;Freq
40  PRINT Freq
50  END
```

# HISTogram:HITS

Command

```
:MEASure:HISTogram:HITS [<source>]
```

The :MEASure:HISTogram:HITS command enables the number of hits within the histogram measurement. The source is specified with the MEASure:SOURce command or with the optional parameter following the HITS command. The HISTogram:HITS measurement only applies to the histogram waveform or memories containing histograms.

The measurement requires that the histogram feature be enabled using the :HISTogram:MODE command.

`<source>` {WMEMory<number> | HISTogram}

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

**Example**

This example measures the number of hits within the histogram stored in WMEMory1.

```
10 OUTPUT 707;"MEASURE:HISTOGRAM:HITS WMEMORY1"
20 END
```

Query

```
:MEASure:HISTogram:HITS? [<source>]
```

The :MEASure:HISTogram:HITS? query returns the number of hits measurement within the histogram.

Returned Format

```
[:MEASure:HISTogram:HITS]<value>[,<result_state>]<NL>
```

<value> The number of hits in the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example returns the number of hits within the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:HITS?"
30 ENTER 707;Histhits
40 PRINT Histhits
50 END
```

# HISTogram:MEAN

Command

```
:MEASure:HISTogram:MEAN [<source>]
```

The :MEASure:HISTogram:MEAN command enables the mean of the histogram measurement. The mean of the histogram is the average value of all the points in the histogram. The source is specified with the MEASure:SOURce command or with the optional parameter following the MEAN command. The HISTogram:MEAN measurement only applies to the histogram waveform or memories containing histograms.

The measurement requires that the histogram feature be enabled using the :HISTogram:MODE command.

`<source>` {WMEMory<number> | HISTogram}

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

**Example**

This example measures the mean of the histogram source.

```
10 OUTPUT 707;"MEASURE:HISTOGRAM:MEAN HISTOGRAM"
20 END
```

Query

```
:MEASure:HISTogram:MEAN? [<source>]
```

The :MEASure:HISTogram:MEAN? query returns the measurement of the mean of the histogram.

Returned Format

```
[:MEASure:HISTogram:MEAN]<value>[,<result_state>]<NL>
```

`<value>` The mean of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example returns the mean of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:MEAN?"
30 ENTER 707;Histmean
40 PRINT Histmean
50 END
```

# HISTogram:MEDian

**Command**

```
:MEASure:HISTogram:MEDian [<source>]
```

The :MEASure:HISTogram:MEDian command enables the median of the histogram measurement. The median of the histogram is the time or voltage of the point at which 50% of the histogram is to the left or right (above or below for vertical histograms). The source is specified with the MEASure:SOURce command or with the optional parameter following the MEDian command. The HISTogram:MEDian measurement only applies to the histogram waveform or memories containing histograms.

The measurement requires that the histogram feature be enabled using the :HISTogram:MODE command.

`<source>` {WMEMory<number> | HISTogram}

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

**Example**

This example measures the median of the histogram whose source has been defined with the MEASure:SOURce command.

```
10 OUTPUT 707;"MEASURE:HISTOGRAM:MEDIAN"
20 END
```

Query

```
:MEASure:HISTogram:MEDian? [<source>]
```

The :MEASure:HISTogram:MEDian? query returns the measurement of the median of the histogram.

Returned Format

```
[:MEASure:HISTogram:MEDian]<value>[,<result_state>]<NL>
```

<value> The median of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example returns the median of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:MEDIAN?"
30 ENTER 707;Histmed
40 PRINT Histmed
50 END
```

# HISTogram:M1S

**Command**

```
:MEASure:HISTogram:M1S [<source>]
```

The :MEASure:HISTogram:M1S command enables the percentage of points measurement that are within one standard deviation of the mean of the histogram. The source is specified with the MEASure:SOURce command or with the optional parameter following the M1S command. The HISTogram:M1S measurement only applies to the histogram waveform or memories containing histograms.

The measurement requires that the histogram feature be enabled using the :HISTogram:MODE command.

`<source>` {WMEMory<number> | HISTogram}

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

**Example**

This example measures the percentage of points that are within one standard deviation of the mean of the histogram of the data stored in waveform memory 3.

```
10 OUTPUT 707;"MEASURE:HISTOGRAM:M1S WMEMORY3"
20 END
```

Query

```
:MEASure:HISTogram:M1S? [<source>]
```

The :MEASure:HISTogram:M1S? query returns the measurement of the percentage of points within one standard deviation of the mean of the histogram.

Returned Format

```
[:MEASure:HISTogram:M1S]<value>[,<result_state>]<NL>
```

`<value>` The percentage of points within one standard deviation of the mean of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example returns the percentage of points within one standard deviation of the mean of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:M1S?"
30 ENTER 707;Histm1s
40 PRINT Histm1s
50 END
```

# HISTogram:M2S

**Command**

```
:MEASure:HISTogram:M2S [<source>]
```

The :MEASure:HISTogram:M2S command enables the percentage of points measurement that are within two standard deviations of the mean of the histogram. The source is specified with the MEASure:SOURce command or with the optional parameter following the M2S command. The HISTogram:M2S measurement only applies to the histogram waveform or memories containing histograms.

The measurement requires that the histogram feature be enabled using the :HISTogram:MODE command.

`<source>` {WMEMory<number> | HISTogram}

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

**Example**

This example measures the percentage of points that are within two standard deviations of the mean of the histogram whose source is specified using the MEASure:SOURce command.

```
10 OUTPUT 707;"MEASURE:HISTOGRAM:M2S"
20 END
```

Query

```
:MEASure:HISTogram:M2S? [<source>]
```

The :MEASure:HISTogram:M2S? query returns the measurement of the percentage of points within two standard deviations of the mean of the histogram.

Returned Format

```
[:MEASure:HISTogram:M2S]<value>[,<result_state>]<NL>
```

`<value>` The percentage of points within two standard deviations of the mean of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example returns the percentage of points within two standard deviations of the mean of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:M2S?"
30 ENTER 707;Histm2s
40 PRINT Histm2s
50 END
```

# HISTogram:M3S

**Command**

```
:MEASure:HISTogram:M3S [<source>]
```

The :MEASure:HISTogram:M3S command enables the percentage of points measurement that are within three standard deviations of the mean of the histogram. The source is specified with the MEASure:SOURce command or with the optional parameter following the M3S command. The HISTogram:M3S measurement only applies to the histogram waveform or memories containing histograms.

The measurement requires that the histogram feature be enabled using the :HISTogram:MODE command.

`<source>` {WMEMory<number> | HISTogram}

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

**Example**

This example measures the percentage of points that are within three standard deviations of the mean of the histogram.

```
10 OUTPUT 707;"MEASURE:HISTOGRAM:M3S HISTOGRAM"
20 END
```

Query

```
:MEASure:HISTogram:M3S? [<source>]
```

The :MEASure:HISTogram:M3S? query returns the measurement of the percentage of points within three standard deviations of the mean of the histogram.

Returned Format

```
[:MEASure:HISTogram:M3S]<value>[,<result_state>]<NL>
```

`<value>` The percentage of points within three standard deviations of the mean of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example returns the percentage of points within three standard deviations of the mean of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"     !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:M3S?"
30 ENTER 707;Histm3s
40 PRINT Histm3s
50 END
```

# HISTogram:PEAK

**Command**

```
:MEASure:HISTogram:PEAK [<source>]
```

The MEASure:HISTogram:PEAK command enables the number of hits in the greatest peak of the histogram measurement. The source is specified with the MEASure:SOURce command or with the optional parameter following the PEAK command. The HISTogram:PEAK measurement only applies to the histogram waveform or memories containing histograms.

The measurement requires that the histogram feature be enabled using the :HISTogram:MODE command.

`<source>` {WMEMory<number> | HISTogram}

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

**Example**

This example measures the number of hits in the greatest peak of the histogram stored in waveform memory 2.

```
10 OUTPUT 707;"MEASURE:HISTOGRAM:PEAK WMEMORY2"
20 END
```

Query

```
:MEASure:HISTogram:PEAK? [<source>]
```

The :MEASure:HISTogram:PEAK? query returns the number of hits in the greatest peak of the histogram measurement.

Returned Format

```
[:MEASure:HISTogram:PEAK]<value>[,<result_state>]<NL>
```

`<value>` The number of hits in the histogram peak.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example returns the number of hits in the greatest peak of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:PEAK?"
30 ENTER 707;Histpeak
40 PRINT Histpeak
50 END
```

# HISTogram:PP

Command

```
:MEASure:HISTogram:PP [<source>]
```

The MEASure:HISTogram:PP command enables the width measurement of the histogram. The width is measured as the time or voltage of the last histogram bucket with data in it minus the time or voltage of the first histogram bucket with data in it. The source is specified with the MEASure:SOURce command or with the optional parameter following the PP command. The HISTogram:PP measurement only applies to the histogram waveform or memories containing histograms.

The measurement requires that the histogram feature be enabled using the :HISTogram:MODE command.

`<source>` {WMEMory<number> | HISTogram}

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

**Example**

This example measures the width of the histogram.

```
10 OUTPUT 707;"MEASURE:HISTOGRAM:PP HISTOGRAM"
20 END
```

Query

```
:MEASure:HISTogram:PP? [<source>]
```

The :MEASure:HISTogram:PP? query returns the measurement of the width of the histogram.

Returned Format

```
[:MEASure:HISTogram:PP]<value>[,<result_state>]<NL>
```

<value> The width of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example returns the width of the current histogram and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:PP?"
30 ENTER 707;Histpp
40 PRINT Histpp
50 END
```

# HISTogram:STDDev

Command

```
:MEASure:HISTogram:STDDev [<source>]
```

The :MEASure:HISTogram:STDDev command enables the standard deviation of the histogram measurement. The source is specified with the MEASure:SOURce command or with the optional parameter following the STDDev command. The HISTogram:STDDev measurement only applies to the histogram waveform or memories containing histograms.

The measurement requires that the histogram feature be enabled using the :HISTogram:MODE command.

`<source>` {WMEMory<number> | HISTogram}

`<number>` For waveform memories (WMEMory): 1,2,3, or 4.

**Example**

This example measures the standard deviation of the histogram whose source is specified using the MEASure:SOURce command.

```
10 OUTPUT 707;"MEASURE:HISTOGRAM:STDDEV"
20 END
```

Query

```
:MEASure:HISTogram:STDDev? [<source>]
```

The :MEASure:HISTogram:STDDev? query returns the measurement of standard deviation of the histogram.

Returned Format

```
[:MEASure:HISTogram:STDDev]<value>[,<result_state>]<NL>
```

`<value>` The standard deviation of the histogram.

`<result_state>` If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

**Example**

This example returns the standard deviation of the histogram whose source is specified using the MEASure:SOURce command and prints the result to the computer's screen.

```
10 OUTPUT 707;":SYSTEM:HEADER OFF"      !Response headers off
20 OUTPUT 707;":MEASURE:HISTOGRAM:STDDEV?"
30 ENTER 707;Histsttd
40 PRINT Histsttd
50 END
```

# JITTer:DIRection

Command

```
:MEASure:JITTer:DIRection {RISing | FALLing |
FODisplay}
```

The :MEASure:JITTer:DIRection command selects the which edge of your waveform is used when measurement statistics are computed for all complete cycles of a waveform within the waveform viewing area. The :MEASure:JITTer:STATistics must be on before issuing this command. This command only affects the following measurements:

- Cycle-to-cycle Jitter
- Duty Cycle
- Frequency
- Period
- Phase

This command and query are only available on the 54845A and 54846A oscilloscope.

**Example**

This example changes the edge used to make measurements on all cycles of the waveform to the falling edge.

```
10  OUTPUT 707;":MEASURE:JITTER:DIRECTION FALLING"
20  END
```

Query

```
:MEASure:JITTer:DIRection?
```

The :MEASure:JITTer:DIRection? query returns the state of the "Edge Direction" control .

Returned Format

```
[:MEASure:JITTer:DIRection] {RISing | FALLing |
FODisplay}<NL>
```

**Example**

This example places the current state of the "Edge Direction" control in the variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;"MEASURE:JITTER:DIRECTION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# JITTer:STATistics

Command

```
:MEASure:JITTer:STATistics {{OFF | 0} | {ON | 1}}
```

The :MEASure:JITTer:STATistics command enables measurement statistics to be computed for all complete cycles of a waveform within the waveform viewing area. This command only affects the following measurements:

- +width
- -width
- Delta Time
- Duty Cycle
- Fall Time
- Frequency
- Period
- Rise Time

This command are only available on the 54845A and 54846A oscilloscopes.

**Example**

This example enables measurement statistics on all cycles of the waveform.

```
10  OUTPUT 707;":MEASURE:JITTER:STATISTICS ON"
20  END
```

Query

```
:MEASure:JITTer:STATistics?
```

The :MEASure:JITTer:STATistics? query returns the state of the "Compute statistics on all measurements in the waveform" control .

This query are only available on the 54845A and 54846A oscilloscopes.

Returned Format

```
[:MEASure:JITTer:STATistics] {0 | 1}<NL>
```

**Example**

This example places the current state of the "Compute statistics on all measurements in the waveform" control in the variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;"MEASURE:JITTER:STATISTICS?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# NWIDth

**Command**

```
:MEASure:NWIDth [<source>]
```

The :MEASure:NWIDth command measures the width of the first negative pulse on the screen using the mid-threshold levels of the waveform (50% levels with standard measurements selected). Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:NWIDth command.

The algorithm is:

If the first edge on the screen is rising,

then

nwidth = time at the second rising edge – time at the first falling edge

else

nwidth = time at the first rising edge – time at the first falling edge.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the width of the first negative pulse on the screen.

```
10  OUTPUT 707;":MEASURE:NWIDTH"
20  END
```

Query

```
:MEASure:NWIDth? [<source>]
```

The :MEASure:NWIDth? query returns the measured width of the first negative pulse of the specified source.

Returned Format

```
[:MEASure:NWIDth] <value>[,<result_state>]<NL>
```

<value> The width of the first negative pulse on the screen using the mid-threshold levels of the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current width of the first negative pulse on the screen in the numeric variable, Width, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:NWIDTH?"
30  ENTER 707;Width
40  PRINT Width
50  END
```

# OVERshoot

**Command**

```
:MEASure:OVERshoot [<source>]
```

The :MEASure:OVERshoot command measures the overshoot of the first edge on the screen. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:OVERshoot command.

The algorithm is:

If the first edge on the screen is rising,

then

overshoot = (Local Vmax – Vtop) / Vamplitude

else

overshoot = (Vbase – Local Vmin) / Vamplitude.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the overshoot of the first edge on the screen.

```
10  OUTPUT 707;":MEASURE:OVERSHOOT"
20  END
```

Query

```
:MEASure:OVERshoot? [<source>]
```

The :MEASure:OVERshoot? query returns the measured overshoot of the specified source.

Returned Format

```
[:MEASure:OVERshoot] <value>[,<result_state>]<NL>
```

<value> Ratio of overshoot to amplitude, in percent.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current value of overshoot in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:OVERSHOOT?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# PERiod

Command

```
:MEASure:PERiod [<source>]
```

The :MEASure:PERiod command measures the period of the first complete cycle on the screen using the mid-threshold levels of the waveform (50% levels with standard measurements selected). The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:PERiod command.

The algorithm is:

If the first edge on the screen is rising,

then

period = time at the second rising edge – time at the first rising edge

else

period = time at the second falling edge – time at the first falling edge.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the period of the waveform.

```
10  OUTPUT 707;":MEASURE:PERIOD"
20  END
```

Query

```
:MEASure:PERiod? [<source>]
```

The :MEASure:PERiod? query returns the measured period of the specified source.

Returned Format

```
[:MEASure:PERiod] <value>[,<result_state>]<NL>
```

<value> Period of the first complete cycle on the screen.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current period of the waveform in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:PERIOD?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# PHASe

Command

```
:MEASure:PHASe [<source>[,<source>]]
```

The :MEASure:PHASe command measures the phase in degrees between two edges. If one source is specified, the phase from the specified edge of the first source to the specified edge of the second source is measured. If one source is specified, the phase is always 0.0E0.00°.

The edge that is used for the measurement can be changed for the 54845A and 54846A oscilloscopes by using the :MEASure:JITTer:DIRection command. This also requires the :MEASure:JITTer:STATistics to be on. All other Infiniium oscilloscopes use the rising edge for the measurement.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 or 3, for two channel acquisition mode. (54835/45A/46A)
An integer, 1 - 4, for four channel acquisition mode. (54835/45A/46A)

An integer, 1-2, for the 54810/20A oscilloscopes.

An integer, 1-4, for all other Infiniium oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the phase between channel 1 and channel 2.

```
10  OUTPUT 707;":MEASURE:PHASE CHANNEL1,CHANNEL2"
20  END
```

**Related Commands**

:MEASure:JITTer:DIRection

Query

```
:MEASure:PHASe? [<source>[,<source>]]
```

The :MEASure:PHASe? query returns the measured phase angle value.

The necessary waveform edges must be present on the display. The query will return 9.99999E+37 if the necessary edges are not displayed.

Returned Format

```
[:MEASure:PHASe] <value>[,result_state]<NL>
```

<value> Phase angle from the first edge on the first source to the first edge edge on the second source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current phase angle value between channel 1 and channel 2 in the variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:PHASE? CHANNEL1,CHANNEL2"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# PREShoot

Command

```
:MEASure:PREShoot [<source>]
```

The :MEASure:PREShoot command measures the preshoot of the first edge on the screen. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:PREShoot command.

The algorithm is:

If the first edge on the screen is rising,

then

preshoot = (Vbase – Local Vmin) / Vamplitude

else

preshoot = (Local Vmax – Vtop) / Vamplitude.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the preshoot of the waveform on the screen.

```
10  OUTPUT 707;":MEASURE:PRESHOOT"
20  END
```

Query

```
:MEASure:PREShoot?[<source>]
```

The :MEASure:PREShoot? query returns the measured preshoot of the specified source.

Returned Format

```
[:MEASure:PREShoot] <value>[,<result state>]<NL>
```

<value> Ratio of preshoot to amplitude, in percent.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current value of preshoot in the numeric variable, Preshoot, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:PRESHOOT?"
30  ENTER 707;Preshoot
40  PRINT Preshoot
50  END
```

# PWIDth

Command

```
:MEASure:PWIDth [<source>]
```

The :MEASure:PWIDth command measures the width of the first positive pulse on the screen using the mid-threshold levels of the waveform (50% levels with standard measurements selected). Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:PWIDth command.

The algorithm is:

If the first edge on the screen is rising,

then

pwidth = time at the first falling edge – time at the first rising edge

else

pwidth = time at the second falling edge – time at the first rising edge.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the width of the first positive pulse on the screen.

```
10  OUTPUT 707;":MEASURE:PWIDTH"
20  END
```

Query

```
:MEASure:PWIDth?[<source>]
```

The :MEASure:PWIDth? query returns the measured width of the first positive pulse of the specified source.

Returned Format

```
[:MEASure:PWIDth] <value>[,<result_state>]<NL>
```

<value> Width of the first positive pulse on the screen in seconds.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the value of the width of the first positive pulse on the screen in the numeric variable, Width, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:PWIDTH?"
30  ENTER 707;Width
40  PRINT Width
50  END
```

## RESults?

Query

```
:MEASure:RESults?
```

The :MEASure:RESults? query returns the results of the continuous measurements. The measurement results always include only the current results. If SENDvalid is ON, the measurement results state is returned immediately following the measurement result. If :MEASure:STATistics is ON, the measurement results include the current, minimum, maximum, mean, standard deviation, and statistical sample size of each measurement.

If more than one measurement is running continuously, the values in the :MEASure:RESults table in this chapter will be duplicated for each continuous measurement from the first to last (top to bottom) of display. There may a maximum of four continuous measurements at a time.

Returned Format

```
[:MEASure:RESults] <result_list><NL>
```

`<result_list>` A list of the measurement results, as in the :MEASure:RESults table in this chapter, separated with commas.

**Example**

This example places the current results of the measurements in the string variable, Result$, then prints the contents of the variable to the computer's screen.

```
10  DIM Result$[200]!Dimension variable
20  OUTPUT 707;":MEASURE:RESULTS?"
30  ENTER 707;Result$
40  PRINT Result$
50  END
```

**Table 20-2** **Result States**

| Code | Result | Description |
|---|---|---|
| 0 | RESULT_CORRECT | Result correct. No problem found. |
| 1 | RESULT_QUESTIONABLE | Result questionable but could be measured. |
| 2 | RESULT_LESS_EQ | Result less than or equal to value returned. |
| 3 | RESULT_GTR_EQ | Result greater than or equal to value returned. |
| 4 | RESULT_INVALID | Result returned is invalid. |
| 5 | EDGE_NOT_FOUND | Result invalid. Required edge not found. |
| 6 | MAX_NOT_FOUND | Result invalid. Max not found. |
| 7 | MIN_NOT_FOUND | Result invalid. Min not found. |
| 8 | TIME_NOT_FOUND | Result invalid. Requested time not found. |
| 9 | VOLT_NOT_FOUND | Result invalid. Requested voltage not found. |
| 10 | TOP_EQUALS_BASE | Result invalid. Top and base are equal. |
| 11 | MEAS_ZONE_SMALL | Result invalid. Measurement zone too small. |
| 12 | LOWER_INVALID | Result invalid. Lower threshold not on waveform. |
| 13 | UPPER_INVALID | Result invalid. Upper threshold not on waveform. |
| 14 | UPPER_LOWER_INVALID | Result invalid. Upper and lower thresholds are too close. |
| 15 | TOP_INVALID | Result invalid. Top not on waveform. |
| 16 | BASE_INVALID | Result invalid. Base not on waveform. |
| 17 | INCOMPLETE | Result invalid. Completion criteria not reached. |
| 18 | INVALID_SIGNAL | Result invalid. Measurement invalid for this type of waveform. |
| 19 | SIGNAL_NOT_DISPLAYED | Result invalid. waveform is not displayed. |
| 20 | CLIPPED_HIGH | Result invalid. Waveform is clipped high. |
| 21 | CLIPPED_LOW | Result invalid. Waveform is clipped low. |
| 22 | CLIPPED_HIGH_LOW | Result invalid. Waveform is clipped high and low. |
| 23 | ALL_HOLES | Result invalid. Data contains all holes. |
| 24 | NO_DATA | Result invalid. No data on screen. |
| 25 | CURSOR_OFF_SCREEN | Result invalid. Cursor is not on screen. |
| 26 | MEASURE_CANCELED | Result invalid. Measurement aborted. |
| 27 | MEASURE_TIMEOUT | Result invalid. Measurement timed-out. |
| 28 | NO_MEAS | Result invalid. No measurement to track. |
| 29 | PEAK_NOT_FOUND | Result invalid. FFT peak not found. |
| 30 | BAD_EYE | Result invalid. Eye pattern not found. |

| | | |
|---|---|---|
| 31 | BAD_NRZ | Result invalid. No NRZ eye pattern found. |
| 32 | BAD_ER_CAL | Result invalid. No valid extinction Ratio calibration. |
| 33 | NOT_1_CHANNEL | Result invalid. There is more than one source on creating the database. |
| 34 | DONT_DISPLAY_MEAS | Result invalid. Do not display the measurement. |
| 35 | SMALL_SIGNAL | Signal may be too small to evaluate. |
| 36 | RESULT_AVERAGING | Result invalid. Awaiting completion of averaging. |

# RISetime

Command

```
:MEASure:RISetime [<source>]
```

The :MEASure:RISetime command measures the rise time of the first displayed edge by measuring the time at the lower threshold of the rising edge, measuring the time at the upper threshold of the rising edge, then calculating the rise time with the following algorithm:

Rise time = time at upper threshold point – time at lower threshold point.

Sources are specified with the :MEASure:SOURce command or with the optional parameter following the RISetime command. With standard measurements selected, the lower threshold is at the 10% point and the upper threshold is at the 90% point on the rising edge.

<source> {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

<N> CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the rise time of the displayed waveform.

```
10  OUTPUT 707;":MEASURE:RISETIME"
20  END
```

Query `:MEASure:RISetime?[<source>]`

The :MEASure:RISetime? query returns the rise time of the specified source.

Returned Format `[:MEASure:RISetime] <value>[,<result_state>]<NL>`

`<value>` Rise time in seconds.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current value of rise time in the numeric variable, Rise, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:RISETIME?"
30  ENTER 707;Rise
40  PRINT Rise
50  END
```

# SCRatch

Command

```
:MEASure:{SCRatch | CLEar}
```

The :MEASure:SCRatch command clears the measurement results from the screen. This command performs the same function as :MEASure:CLEar.

**Example**

This example clears the current measurement results from the screen.

```
10  OUTPUT 707;":MEASURE:SCRATCH"
20  END
```

# SENDvalid

Command

```
:MEASure:SENDvalid {{OFF|0} | {ON|1}}
```

The :MEASure:SENDvalid command enables the result state code to be returned with the :MEASure:RESults? query.

**Example**

This example turns the send valid function on.

```
10  OUTPUT 707;":MEASURE:SENDVALID ON"
20  END
```

Query

```
:MEASure:SENDvalid?
```

The :MEASure:SENDvalid? query returns the state of the send valid control.

Returned Format

```
{:MEASure:SENDvalid] {0 | 1}<NL>
```

**Example**

This example places the current mode for SENDvalid in the string variable, Mode$, then prints the contents of the variable to the computer's screen.

```
10  DIM Mode$[50]!Dimension variable
20  OUTPUT 707;":MEASURE:SENDVALID?"
30  ENTER 707;Mode$
40  PRINT Mode$
50  END
```

**See Also**

Refer to the :MEASure:RESults? query for information on the results returned and how they are affected by the SENDvalid command. Refer to the individual measurements for information on how the result state is returned.

# SOURce

Command

```
:MEASure:SOURce <source>[,<source>]
```

The :MEASure:SOURce command selects the source for measurements. You can specify one or two sources with this command. All measurements except :MEASure:DELTatime are made on the first specified source. The delta time measurement uses two sources if two are specified. If only one source is specified, the delta time measurement uses that source for both of its parameters.

<source> {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

<N> CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example selects channel 1 as the source for measurements.

```
10  OUTPUT 707;":MEASURE:SOURCE CHANNEL1"
20  END
```

Query

```
:MEASure:SOURce?
```

The :MEASure:SOURce? query returns the current source selection.

Returned Format

```
[:MEASure:SOURce] <source>[,<source>]<NL>
```

**Example**

This example places the currently specified sources in the string variable, Source$, then prints the contents of the variable to the computer's screen.

```
10  DIM Source$[50]!Dimension variable
20  OUTPUT 707;":MEASURE:SOURCE?"
30  ENTER 707;Source$
40  PRINT Source$
50  END
```

## STATistics

Command

```
:MEASure:STATistics {{OFF|0} | {ON|1}}
```

The :MEASure:STATistics command turns the statistics measurements on and off. The statistics state only affects the information returned by the :MEASure:RESults? query.

**Example**

This example turns the statistics function on.

```
10  OUTPUT 707;":MEASURE:STATISTICS ON"
20  END
```

Query

```
:MEASure:STATistics?
```

The :MEASure:STATistics? query returns the current statistics mode.

Returned Format

```
[:MEASure:STATistics] {0 | 1}<NL>
```

**Example**

This example places the current mode for statistics in the string variable, Mode$, then prints the contents of the variable to the computer's screen.

```
10  DIM Mode$[50]!Dimension variable
20  OUTPUT 707;":MEASURE:STATISTICS?"
30  ENTER 707;Mode$
40  PRINT Mode$
50  END
```

**See Also**

Refer to the :MEASure:RESults? query for information on the result returned and how it is affected by the STATistics command.

# TEDGe

Command

```
:MEASure:TEDGe <meas_thres_txt>,
[<slope>]<occurrence>[,<source>]
```

The :MEASure:TEDGe command measures the time interval between the trigger event and the specified edge (threshold level, slope, and transition). Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TEDGe command.

`<meas_thres_txt>` UPPer, MIDDle, or LOWer to identify the threshold.

`<slope>` { - (minus) for falling | + (plus) for rising | <none> (the slope is optional; if no slope is specified, + (plus) is assumed) }

`<occurrence>` An integer value representing the edge of the occurrence. The desired edge must be present on the display. Edges are counted with 1 being the first edge from the left on the display, and a maximum value of 65534.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Query

```
:MEASure:TEDGe? <meas_thres_txt>,
<slope><occurrence> [,<source>]
```

The :MEASure:TEDGe? query returns the time interval between the trigger event and the specified edge (threshold level, slope, and transition).

Returned Format

```
[:MEASure:TEDGe] <time>[,<result_state>]<NL>
```

`<time>` The time interval between the trigger event and the specified voltage level and transition.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example returns the time interval between the trigger event and the 90% threshold on the second rising edge of the source waveform to the numeric variable, Time. The contents of the variable are then printed to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:TEDGE? UPPER,+2"
30  ENTER 707;Time
40  PRINT Time
50  END
```

**Turn Off Headers**

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

# TMAX

Command

```
:MEASure:TMAX [<source>]
```

The :MEASure:TMAX command measures the first time at which the maximum voltage of the source waveform occurred. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TMAX command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Query `:MEASure:TMAX? [<source>]`

The :MEASure:TMAX? query returns the time at which the first maximum voltage occurred.

Returned Format `[:MEASure:TMAX] <time>[,<result_state>]<NL>`

`<time>` Time at which the first maximum voltage occurred or frequency where the maximum FFT amplitude occurred.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example** This example returns the time at which the first maximum voltage occurred to the numeric variable, Time, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:TMAX?"
30  ENTER 707;Time
40  PRINT Time
50  END
```

# TMIN

Command

```
:MEASure:TMIN [<source>]
```

The :MEASure:TMIN command measures the time at which the first minimum voltage occurred. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TMIN command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

Query

```
:MEASure:TMIN? [<source>]
```

The :MEASure:TMIN? query returns the time at which the first minimum voltage occurred or the frequency where the minimum FFT amplitude occurred.

Returned Format

```
[:MEASure:TMIN] <time>[,<result_state>]<NL>
```

`<time>` Time at which the first minimum voltage occurred.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example returns the time at which the first minimum voltage occurred to the numeric variable, Time, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:TMIN?"
30  ENTER 707;Time
40  PRINT Time
50  END
```

# TVOLt

**Command**

```
:MEASure:TVOLt <voltage>,[<slope>]<occurrence>
[,<source>]
```

The :MEASure:TVOLt command measures the time interval between the trigger event and the defined voltage level and transition. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TVOLt command.

**Query**

```
:MEASure:TVOLt? <voltage>,<slope><occurrence>
[,<source>]
```

The :MEASure:TVOLt? query returns the time interval between the trigger event and the specified voltage level and transition.

`<voltage>` Voltage level at which time will be measured.

`<slope>` The direction of the waveform change when the specified voltage is crossed - rising (+) or falling (–). If no +/- sign is present, + is assumed.

`<occurrence>` The number of the crossing to be reported (if one, the first crossing is reported: if two, the second crossing is reported, etc.). The desired edge (crossing) must be present on the display. Edges are counted with 1 being the first edge from the left of the display, and a maximum value of 65534.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Returned Format**

```
[:MEASure:TVOLt] <time>[,<result_state>]<NL>
```

`<time>` The time interval between the trigger event and the specified voltage level and transition.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example returns the time interval between the trigger event and the transition through –.250 Volts on the third rising edge of the source waveform to the numeric variable, Time. The contents of the variable are then printed to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:TVOLT? -.250,+3"
30  ENTER 707;Time
40  PRINT Time
50  END
```

# VAMPlitude

Command

```
:MEASure:VAMPlitude [<source>]
```

The :MEASure:VAMPlitude command calculates the difference between the top and base voltage of the specified source. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VAMPlitude command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example calculates the difference between the top and base voltage of the specified source.

```
10  OUTPUT 707;":MEASURE:VAMPLITUDE"
20  END
```

Query

```
:MEASure:VAMPlitude? [<source>]
```

The :MEASure:VAMPlitude? query returns the calculated difference between the top and base voltage of the specified source.

Returned Format

```
[:MEASure:VAMPlitude] <value>[,<result_state>]<NL>
```

`<value>` Calculated difference between the top and base voltage.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current Vamplitude value in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VAMPLITUDE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# VAVerage

**Command**

```
:MEASure:VAVerage {CYCLe | DISPlay}[,<source>]
```

The :MEASure:VAVerage command calculates the average voltage over the displayed waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VAVerage command.

`CYCLe` The CYCLe parameter instructs the average measurement to measure the average voltage across the first period on the display.

`DISPlay` The DISPlay parameter instructs the average measurement to measure all the data on the display.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example calculates the average voltage over the displayed waveform.

```
10  OUTPUT 707;":MEASURE:VAVERAGE DISPLAY"
20  END
```

Query

```
:MEASure:VAVerage? {CYCLe | DISPlay}[,<source>]
```

The :MEASure:VAVerage? query returns the calculated average voltage of the specified source. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VAVerage command.

Returned Format

```
[:MEASure:VAVerage] <value>[,<result_state>]<NL>
```

`<value>` The calculated average voltage.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current average voltage in the numeric variable, Average, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VAVERAGE? DISPLAY"
30  ENTER 707;Average
40  PRINT Average
50  END
```

# VBASe

**Command**

```
:MEASure:VBASe [<source>]
```

The :MEASure:VBASe command measures the statistical base of the waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VBASe command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the voltage at the base of the waveform.

```
10  OUTPUT 707;":MEASURE:VBASE"
20  END
```

Query

```
:MEASure:VBASe? [<source>]
```

The :MEASure:VBASe? query returns the measured voltage value at the base of the specified source.

Returned Format

```
[:MEASure:VBASe] <value>[,<result_state>]<NL>
```

`<value>` Voltage at the base of the waveform.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example returns the current voltage at the base of the waveform to the numeric variable, Voltage, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VBASE?"
30  ENTER 707;Voltage
40  PRINT Voltage
50  END
```

# VLOWer

**Command** `:MEASure:VLOWer [<source>]`

The :MEASure:VLOWer command measures the voltage value at the lower threshold of the waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VLOWer command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Query** `:MEASure:VLOWer?`

The :MEASure:VLOWer? query returns the measured lower threshold of the selected source.

**Returned Format** `[:MEASure:VLOWer] <value>[,<result_state>]<NL>`

`<value>` Voltage value at the lower threshold.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example returns the measured voltage at the lower threshold of the waveform to the numeric variable, Vlower, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VLOW?"
30  ENTER 707;Vlower
40  PRINT Vlower
50  END
```

# VMAX

Command

```
:MEASure:VMAX [<source>]
```

The :MEASure:VMAX command measures the absolute maximum voltage present on the selected source waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VMAX command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the absolute maximum voltage on the waveform.

```
10  OUTPUT 707;":MEASURE:VMAX"
20  END
```

Query `:MEASure:VMAX? [<source>]`

The :MEASure:VMAX? query returns the measured absolute maximum voltage or maximum FFT amplitude present on the selected source waveform.

Returned Format `[:MEASure:VMAX] <value>[,<result_state>]<NL>`

`<value>` Absolute maximum voltage present on the waveform.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example returns the measured absolute maximum voltage on the waveform to the numeric variable, Maximum, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VMAX?"
30  ENTER 707;Maximum
40  PRINT Maximum
50  END
```

# VMIDdle

**Command**

```
:MEASure:VMIDdle [<source>]
```

The :MEASure:VMIDdle command measures the voltage level at the middle threshold of the waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VMIDdle command.

**Query**

```
:MEASure:VMIDdle? [<source>]
```

The :MEASure:VMIDdle? query returns the voltage value at the middle threshold of the waveform.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Returned Format**

```
[MEASure:VMIDdle] <value>[,<result_state>]<NL>
```

`<value>` The middle voltage present on the waveform.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example returns the measured middle voltage on the waveform to the numeric variable, Middle, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VMID?"
30  ENTER 707;Middle
40  PRINT Middle
50  END
```

# VMIN

Command

```
:MEASure:VMIN [<source>]
```

The :MEASure:VMIN command measures the absolute minimum voltage present on the selected source waveform. Sources are specified with :MEASure:SOURce or with the optional parameter following the :MEASure:VMIN command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the absolute minimum voltage on the waveform.

```
10  OUTPUT 707;":MEASURE:VMIN"
20  END
```

Query

```
:MEASure:VMIN? [<source>]
```

The :MEASure:VMIN? query returns the measured absolute minimum voltage or minimum FFT amplitude present on the selected source waveform.

Returned Format

```
[:MEASure:VMIN] <value>[,<result_state>]<NL>
```

<value> Absolute minimum voltage present on the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example returns the measured absolute minimum voltage on the waveform to the numeric variable, Minimum, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VMIN?"
30  ENTER 707;Minimum
40  PRINT Minimum
50  END
```

# VPP

**Command**

```
:MEASure:VPP [<source>]
```

The :MEASure:VPP command measures the maximum and minimum voltages on the selected source, then calculates the peak-to-peak voltage as the difference between the two voltages. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VPP command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the peak-to-peak voltage or FFT amplitude range of the previously selected source.

```
10  OUTPUT 707;":MEASURE:VPP"
20  END
```

Query

```
:MEASure:VPP? [<source>]
```

The :MEASure:VPP? query returns the specified source peak-to-peak voltage.

Returned Format

```
[:MEASure:VPP] <value>[,<result_state>]<NL>
```

`<value>` Peak-to-peak voltage of the selected source.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current peak-to-peak voltage in the numeric variable, Voltage, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VPP?"
30  ENTER 707;Voltage
40  PRINT Voltage
50  END
```

# VRMS

**Command**

```
:MEASure:VRMS {CYCLe | DISPlay},{AC | DC} [,<source>]
```

The :MEASure:VRMS command measures the RMS voltage of the selected waveform by subtracting the average value of the waveform from each data point on the display. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VRMS command.

`CYCLe` The CYCLe parameter instructs the RMS measurement to measure the RMS voltage across the first period of the display.

`DISPlay` The DISPLay parameter instructs the RMS measurement to measure all the data on the display. Generally, RMS voltage is measured across one waveform or cycle, however, measuring multiple cycles may be accomplished with the DISPLay option. The DISPlay parameter is also useful when measuring noise.

`AC` The AC parameter is used to measure the RMS voltage subtracting the DC component.

`DC` The DC parameter is used to measure RMS voltage including the DC component.

The AC RMS, DC RMS, and VAVG parameters are related as in this formula:

$DCVRMS^2 = ACVRMS^2 + VAVG^2$

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the RMS voltage of the previously selected waveform.

```
10  OUTPUT 707;":MEASURE:VRMS CYCLE,AC"
20  END
```

Query

```
:MEASure:VRMS? {CYCLe | DISplay},{AC | DC}
[,<source>]
```

The :MEASure:VRMS? query returns the RMS voltage of the specified source.

Returned Format

```
[:MEASure:VRMS] <value>[,<result_state>]<NL>
```

<value> RMS voltage of the selected waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the current AC RMS voltage over one period of the waveform in the numeric variable, Voltage, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VRMS? CYCLE,AC"
30  ENTER 707;Voltage
40  PRINT Voltage
50  END
```

# VTIMe

**Command**

```
:MEASure:VTIMe <time>[,<source>]
```

The :MEASure:VTIMe command measures the voltage at the specified time. The time is referenced to the trigger event and must be on the screen. When an FFT function is the specified source, the amplitude at the specified frequency is measured. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VTIMe command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

`<time>` A real number for time from trigger in seconds, or frequency in Hertz for an FFT (when a function is set to FFT or a waveform memory contains an FFT).

**Query**

```
:MEASure:VTIMe? <time>[,<source>]
```

The :MEASure:VTIMe? query returns the measured voltage or amplitude.

**Returned Format**

```
[:MEASure:VTIMe] <value>[,<result_state>]<NL>
```

`<value>` Voltage at the specified time. When the source is an FFT function, the returned value is the vertical value at the horizontal setting passed in the VTIMe <time> parameter. The time parameter is in Hertz when an FFT function is the source.

`<result_state>` If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the voltage at 500 ms in the numeric variable, Value, then prints the contents to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VTIME? 500E-3"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# VTOP

Command

```
:MEASure:VTOP [<source>]
```

The :MEASure:VTOP command measures the statistical top of the selected source waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VTOP command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the voltage at the top of the waveform.

```
10  OUTPUT 707;":MEASURE:VTOP"
20  END
```

Query

```
:MEASure:VTOP? [<source>]
```

The :MEASure:VTOP? query returns the measured voltage at the top of the specified source.

Returned Format

```
[:MEASure:VTOP] <value>[,<result_state>]<NL>
```

<value> Voltage at the top of the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the value of the voltage at the top of the waveform in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VTOP?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# VUPPer

Command

```
:MEASure:VUPPer [<source>]
```

The :MEASure:VUPPer command measures the voltage value at the upper threshold of the waveform. Sources are specified with the MEASure:SOURce command or with the optional parameter following the :MEASure:VUPPer command.

`<source>` {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

`<N>` CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

An integer, 1 - 4, representing the selected function or waveform memory.

**Example**

This example measures the voltage at the upper threshold of the waveform.

```
10  OUTPUT 707;":MEASURE:VUPPer"
20  END
```

Query

```
:MEASure:VUPPer? [<source>]
```

The :MEASure:VUPPer? query returns the measured upper threshold value of the selected source.

Returned Format

```
[:MEASure:VUPPer] <value>[,<result_state>]<NL>
```

<value> Voltage at the upper threshold.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

**Example**

This example places the value of the voltage at the upper threshold of the waveform in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":MEASURE:VUPPER?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

21

# Mask Test Commands

# Mask Test Commands

The MTESt subsystem commands and queries control the mask test features. Mask Testing automatically compares measurement results with the boundaries of a set of polygons that you define. Any waveform or sample that falls within the boundaries of one or more polygons is recorded as a failure.

These MTESt commands and queries are implemented in the HP Infiniium Oscilloscopes:

- ALIGn
- AlignFIT
- AMASk:CREate
- AMASk:SOURce
- AMASk:SAVE | STORe
- AMASk:UNITs
- AMASk:XDELta
- AMASk:YDELta
- AUTO
- AVERage
- AVERage:COUNt
- COUNt:FAILures?
- COUNt:FWAVeforms?
- COUNt:WAVeforms?
- DELete
- ENABle
- HAMPlitude
- IMPedance
- INVert
- LAMPlitude
- LOAD
- NREGions?
- PROBe:IMPedance?

- RUMode
- RUMode:SOFailure
- SCALe:BIND
- SCALe:X1
- SCALe:XDELta
- SCALe:Y1
- SCALe:Y2
- SOURce
- STARt | STOP
- STIMe
- TITLe?

# ALIGn

Command

```
:MTESt:ALIGn
```

The :MTESt:ALIGn command automatically aligns and scales the mask to the current waveform on the display. The type of mask alignment performed depends on the current setting of the Use File Setup When Aligning control. See the :MTESt:AUTO command for more information.

**Example**

This example aligns the current mask to the current waveform.

```
10 Output 707;":MTEST:ALIGN"
20 END
```

# AlignFIT

Command

```
:MTESt:AlignFIT {EYEAMI | EYECMI | EYENRZ | FANWidth
| FAPeriod | FAPWidth | FYNWidth | FYPWidth | NONE
| NWIDth | PWIDth | TMAX | TMIN}
```

The :MTESt:AlignFIT command specifies the alignment type for aligning a mask to a waveform. The pulse mask standard has rules that determine which controls the oscilloscope can adjust or change during the alignment process. An X in a column indicates that the control can be adjusted for each of the alignment types of Table 21-1.

**Table 21-1** **Available Alignment Types**

| Alignment Type | Waveform Type | Horizontal Position | 0 Level Voltage | 1 Level Voltage | Vertical Offset | Invert Waveform |
|---|---|---|---|---|---|---|
| EYEAMI | AMI | X | X | X | | |
| EYECMI | CMI | X | X | X | | |
| EYENRZ | NRZ | X | X | X | | |
| FANWidth | Negative | X | | | X | X |
| FAPeriod | Full Period | X | X | | | |
| FAPWidth | Positive | X | | | X | X |
| FYNWidth | Negative | X | X | | | X |
| FYPWidth | Positive | X | X | | | X |
| NONE | Automask | | | | | |
| NWIDth | Negative Pulse | X | X | X | | X |
| PWIDth | Positive Pulse | X | X | X | | X |
| TMAX | Positive Sine Pulse | X | X | X | | X |
| TMIN | Negative Sine Pulse | X | X | X | | X |

**Example**

This example specifies the alignment type to be EYEAMI.

```
10  Output 707;":MTEST:ALIGNFIT EYEAMI"
20  END
```

Query

```
:MTESt:AlignFIT?
```

The :MTEST:AlignFIT? query returns the alignment type used for the mask.

Returned Format

```
[:MTESt:AlignFIT] {EYEAMI | EYECMI | EYENRZ |
FANWidth | FAPeriod | FAPWidth | FYNWidth |
FYPWidth | NONE | NWIDth | PWIDth | TMAX | TMIN}<NL>
```

# AMASk:CREate

Command

```
:MTESt:AMASk:CREate
```

The :MTESt:AMASk:CREate command automatically constructs a mask around the current selected channel, using the tolerance parameters defined by the AMASk:XDELta, AMASk:YDELta, and AMASk:UNITs commands. The mask only encompasses the portion of the waveform visible on the display, so you must ensure that the waveform is acquired and displayed consistently to obtain repeatable results.

The :MTESt:SOURce command selects the channel and should be set before using this command.

**Example**

This example creates an automask using the current XDELta and YDELta units settings.

```
10 OUTPUT 707;":MTEST:AMASK:CREATE"
20 END
```

# AMASk:SOURce

Command

```
:MTESt:AMASk:SOURce CHANnel<number>
```

The :MTESt:AMASk:SOURce command selects the source for the interpretation of the AMASk:XDELta and AMASk:YDELta parameters when AMASk:UNITs is set to CURRent. When UNITs are CURRent, the XDELta and YDELta parameters are defined in terms of the channel units, as set by the :CHANnel:UNITs command, of the selected source. Suppose that UNITs are CURRent and that you set SOURce to CHANNEL1, which is using units of volts. Then you can define AMASk:XDELta in terms of volts and AMASk:YDELta in terms of seconds.

`<number>` An integer, 1 through 4 for the 54815A, 54825A, and 54845A

An integer, 1 through 2 for the 54810A and 54820A

**Example**

This example sets the automask source to Channel 1.

```
10 OUTPUT 707;"MTEST:AMASK:SOURCE CHANNEL1"
20 END
```

Query

```
:MTESt:AMASk:SOURce?
```

The :MTESt:AMASk:SOURce? query returns the currently set source.

Returned Format

```
[:MTESt:AMASk:SOURce] CHANnel<number><NL>
```

**Example**

This example gets the source setting for automask and prints the result on the computer display.

```
10 DIM Amask_source$[30]
20 OUTPUT 707;"MTEST:AMASK:SOURCE?"
30 ENTER 707;Amask_source$
40 PRINT Amask_source$
50 END
```

# AMASk:SAVE | STORe

**Command**

```
:MTESt:AMASk:SAVE|STORe "<filename>"
```

The :MTESt:AMASk:SAVE command saves the automask generated mask to a file. If an automask has not been generated, an error occurs.

`<filename>` An MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used). The filename assumes the present working directory if a path does not precede the file name.

**Example**

This example saves the automask generated mask to a file named "FILE1".

```
10 OUTPUT 707;":MTEST:AMASK:SAVE""FILE1"""
20 END
```

# AMASk:UNITs

Command

```
:MTESt:AMASk:UNITs {CURRent | DIVisions}
```

The :MTESt:AMASk:UNITs command alters the way the mask test subsystem interprets the tolerance parameters for automasking as defined by AMASk:XDELta and AMASk:YDELta commands.

`CURRent` When set to CURRent, the mask test subsystem uses the units as set by the :CHANnel:UNITs command, usually time for ΔX and voltage for ΔY.

`DIVisions` When set to DIVisions, the mask test subsystem uses the graticule as the measurement system, so tolerance settings are specified as parts of a screen division. The mask test subsystem maintains separate XDELta and YDELta settings for CURRent and DIVisions. Thus, XDELta and YDELta are not converted to new values when the UNITs setting is changed.

**Example** This example sets the measurement units for automasking to the current :CHANnel:UNITs setting.

```
10 OUTPUT 707;"MTEST:AMASK:UNITS CURRENT"
20 END
```

Query

```
:MTESt:AMASk:UNITs?
```

The AMASk:UNITs query returns the current measurement units setting for the mask test automask feature.

Returned Format

```
[:MTESt:AMASk:UNITs] {CURRent | DIVision}<NL>
```

**Example** This example gets the automask units setting, then prints the setting on the screen of the computer.

```
10 DIM Automask_units$[10]
20 OUTPUT 707;"MTEST:AMASK:UNITS?"
30 ENTER 707;Automask_units$
40 PRINT Automask_units$
50 END
```

# AMASk:XDELta

**Command**

```
:MTESt:AMASk:XDELta <xdelta_value>
```

The :MTESt:AMASk:XDELta command sets the tolerance in the X direction around the waveform for the automasking feature. The absolute value of the tolerance will be added and subtracted to horizontal values of the waveform to determine the boundaries of the mask.

`<xdelta_value>` A value for the horizontal tolerance. This value is interpreted based on the setting specified by the AMASk:UNITs command; thus, if you specify 250-E3, the setting for AMASk:UNITs is CURRent, and the current setting specifies time in the horizontal direction, the tolerance will be ±250 ms. If the setting for AMASk:UNITs is DIVisions, the same xdelta_value will set the tolerance to ±250 millidivisions, or 1/4 of a division.

**Example**

This example sets the units to divisions and sets the ΔX tolerance to one-eighth of a division.

```
10 OUTPUT 707;"MTEST:AMASK:UNITS DIVISIONS"
20 OUTPUT 707;":MTEST:AMASK:XDELTA 125E-3"
30 END
```

Query

```
:MTESt:AMASk:XDELta?
```

The AMASk:XDELta? query returns the current setting of the ΔX tolerance for automasking. If your computer program will interpret this value, it should also request the current measurement system using the AMASk:UNITs query.

Returned Format

```
[:MTESt:AMASk:XDELta] <xdelta_value><NL>
```

**Example**

This example gets the measurement system units and ΔX settings for automasking from the oscilloscope and prints the results on the computer screen.

```
10 DIM Automask_units$[10]
20 DIM Automask_xdelta$[20]
30 OUTPUT 707;"MTEST:AMASK:UNITS?"
40 ENTER 707;Automask_units$
50 OUTPUT 707;":MTEST:AMASK:XDELTA?"
60 ENTER 707;Automask_xdelta$
70 PRINT Automask_units$
80 PRINT Automask_xdelta$
90 END
```

# AMASk:YDELta

**Command**

```
:MTESt:AMASk:YDELta <ydelta_value>
```

The :MTESt:AMASk:YDELta command sets the vertical tolerance around the waveform for the automasking feature. The absolute value of the tolerance will be added and subtracted to vertical values of the waveform to determine the boundaries of the mask.

This command requires that mask testing be enabled, otherwise a settings conflict error message is displayed. See :MTESt:ENABle for information on enabling mask testing.

`<ydelta_value>` A value for the vertical tolerance. This value is interpreted based on the setting specified by the AMASk:UNITs command; thus, if you specify 250-E3, the setting for AMASk:UNITs is CURRent, and the current setting specifies voltage in the vertical direction, the tolerance will be ±250 mV. If the setting for AMASk:UNITs is DIVisions, the same ydelta_value will set the tolerance to ±250 millidivisions, or 1/4 of a division.

**Example**

This example sets the units to current and sets the ΔY tolerance to 30 mV, assuming that the current setting specifies volts in the vertical direction.

```
10 OUTPUT 707;"MTEST:AMASK:UNITS CURRENT"
20 OUTPUT 707;":MTEST:AMASK:YDELTA 30E-3"
30 END
```

Query

```
:MTESt:AMASk:YDELta?
```

The AMASk:YDELta? query returns the current setting of the ΔY tolerance for automasking. If your computer program will interpret this value, it should also request the current measurement system using the AMASk:UNITs query.

Returned Format

```
[:MTESt:AMASk:YDELta] <ydelta_value><NL>
```

**Example**

This example gets the measurement system units and ΔY settings for automasking from the oscilloscope and prints the results on the computer screen.

```
10 DIM Automask_units$[10]
20 DIM Automask_ydelta$[20]
30 OUTPUT 707;"MTEST:AMASK:UNITS?"
40 ENTER 707;Automask_units$
50 OUTPUT 707;":MTEST:AMASK:YDELTA?"
60 ENTER 707;Automask_ydelta$
70 PRINT Automask_units$
80 PRINT Automask_ydelta$
90 END
```

# AUTO

**Command**

```
:MTESt:AUTO {{ON|1} | {OFF|0}}
```

The :MTESt:AUTO command enables (ON) or disables (OFF) the Use File Setup When Aligning control. This determines which type of mask alignment is performed when the :MTESt:ALIGn command is sent. When enabled, the oscilloscope controls are changed to the values which are determined by the loaded mask file. This alignment guarantees that the aligned mask and any subsequent mask tests meet the requirements of the standard.

When disabled, the alignment is performed using the current oscilloscope settings. This may be useful when troubleshooting problems during the design phase of a project.

**Example**

This example enables the Use File Settings When Aligning control.

```
10 OUTPUT 707;"MTEST:AUTO ON"
20 END
```

**Query**

```
:MTESt:AUTO?
```

The :MTESt:AUTO? query returns the current value of the Use File Setup When Aligning control.

**Returned Format**

```
[:MTESt:AUTO] {1|0} <NL>
```

**Example**

```
10  OUTPUT 707;":MTEST:AUTO?"
20  ENTER 707;Value
30  PRINT Value
40  END
```

# AVERage

Command

```
:MTESt:AVERage {{ON|1} | {OFF|0}}
```

The :MTESt:AVERage command enables or disables averaging. When ON, the oscilloscope acquires multiple data values for each time bucket, and averages them. When OFF, averaging is disabled. To set the number of averages, use the :MTESt:AVERage:COUNt command described next.

The :ACQuire:AVERage command performs the same function as this command.

Averaging is not available in PDETect mode.

**Example**

This example turns averaging on.

```
10 OUTPUT 707;"MTEST:AVERAGE ON"
20 END
```

Query

```
:MTESt:AVERage?
```

The :MTESt:AVERage? query returns the current setting for averaging.

Returned Format

```
[:MTESt:AVERage] {1|0} <NL>
```

**Example**

This example places the current settings for averaging into the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]   !Dimension variable
20  OUTPUT 707;"MTEST:AVERAGE?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# AVERage:COUNt

Command

```
:MTESt:AVERage:COUNt <count_value>
```

The :MTESt:AVERage:COUNt command sets the number of averages for the waveforms. In the AVERage mode, the :MTESt:AVERage:COUNt command specifies the number of data values to be averaged for each time bucket before the acquisition is considered complete for that time bucket.

The :ACQuire:AVERage:COUNt command performs the same function as this command.

`<count_value>` An integer, 2 to 4096, specifying the number of data values to be averaged.

**Example**

This example specifies that 16 data values must be averaged for each time bucket to be considered complete. The number of time buckets that must be complete for the acquisition to be considered complete is specified by the :MTESt:COMPlete command.

```
10  OUTPUT 707;":MTESt:COUNT 16"
20  END
```

Query

```
:MTESt:COUNt?
```

The :MTESt:COUNt? query returns the currently selected count value.

Returned Format

```
[:MTESt:COUNt] <value><NL>
```

`<value>` An integer, 2 to 4096, specifying the number of data values to be averaged.

**Example**

This example checks the currently selected count value and places that value in the string variable, Result$. The program then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":MTEST:AVERAGE:COUNT?"
30  ENTER 707;Result
40  PRINT Result
50  END
```

# COUNt:FAILures?

Query

```
:MTESt:COUNt:FAILures? REGion<number>
```

The MTESt:COUNt:FAILures? query returns the number of failures that occurred within a particular mask region.

The value 9.999E37 is returned if mask testing is not enabled or if you specify a region number that is unused.

`<number>` An integer, 1 through 8, designating the region for which you want to determine the failure count.

Returned Format

```
[:MTESt:COUNt:FAILures] REGion<number><number_of_failures>
<NL>
```

`<number_of_failures>` The number of failures that have occurred for the designated region.

**Example**

This example determines the current failure count for region 3 and prints it on the computer screen.

```
10 DIM Mask_failures$[50]
20 OUTPUT 707;"MTEST:COUNT:FAILURES? REGION3"
30 ENTER 707;Mask_failures$
40 PRINT Mask_failures$
50 END
```

## COUNt:FWAVeforms?

**Query**

```
:MTESt:COUNt:FWAVeforms?
```

The :MTESt:COUNt:FWAVeforms? query returns the total number of failed waveforms in the current mask test run. This count is for all regions and all waveforms, so if you wish to determine failures by region number, use the COUNt:FAILures? query.

This count may not always be available. It is available only when the following conditions are true:

- Mask testing was turned on before the histogram or color grade persistence, and
- No mask changes have occurred, including scaling changes, editing, or new masks.

The value 9.999E37 is returned if mask testing is not enabled, or if you have modified the mask.

**Returned Format**

```
[:MTESt:COUNt:FWAVeforms] <number_of_failed_waveforms><NL>
```

`<number_of_failed_waveforms>` The total number of failed waveforms for the current test run.

**Example**

This example determines the number of failed waveforms and prints the result on the computer screen.

```
10 OUTPUT 707;"SYSTEM:HEADER OFF"
20 OUTPUT 707;":MTEST:COUNT:FWAVEFORMS?
30 ENTER 707;Mask_fwaveforms$
40 PRINT Mask_fwaveforms$
50 END
```

# COUNt:WAVeforms?

Query

```
:MTESt:COUNt:WAVeforms?
```

The :MTESt:COUNt:WAVeforms? query returns the total number of waveforms acquired in the current mask test run. The value 9.999E37 is returned if mask testing is not enabled.

Returned Format

```
[:MTESt:COUNt:WAVeforms] <number_of_waveforms><NL>
```

<number_of_waveforms> The total number of waveforms for the current test run.

**Example**

This example determines the number of waveforms acquired in the current test run and prints the result on the computer screen.

```
10 OUTPUT 707;"SYSTEM:HEADER OFF"
20 OUTPUT 707;":MTEST:COUNT:WAVEFORMS?"
30 ENTER 707;Mask_waveforms
40 PRINT Mask_waveforms
50 END
```

# DELete

Command

```
:MTESt:DELete
```

The :MTESt:DELete command clears the currently loaded mask.

**Example**

This example clears the currently loaded mask.

```
10 OUTPUT 707;"MTEST:DELETE"
20 END
```

## ENABle

Command

```
:MTESt:ENABle {{ON|1} | {OFF|0}}
```

The :MTESt:ENABle command enables or disables the mask test features.

ON Enables the mask test features.

OFF Disables the mask test features.

**Example**

This example enables the mask test features.

```
10 OUTPUT 707;":MTEST:ENABLE ON"
20 END
```

Query

```
:MTESt:ENABle?
```

The :MTESt:ENABle? query returns the current state of mask test features.

Returned Format

```
[MTESt:ENABle] {1|0}<NL>
```

**Example**

This example places the current value of the mask test state in the numeric variable Value, then prints the contents to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF
20  OUTPUT 707;":MTEST:ENABLE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# HAMPlitude

Command

```
:MTESt:HAMPlitude <upper_limit>
```

The :MTESt:HAMPlitude command sets the maximum pulse amplitude value that passes the pulse standard. For some of the pulse communications standards, a pulse has a range of amplitude values and still passes the standard. This command sets the upper limit used during mask testing.

`<upper_limit>` A real number that represents the maximum amplitude in volts of a pulse as allowed by the pulse standard.

**Example**

This example sets the maximum pulse amplitude to 3.6 volts.

```
10  OUTPUT 707;"MTEST:HAMPLITUDE 3.6"
20  END
```

Query

```
:MTESt:HAMPlitude?
```

The :MTESt:HAMPlitude? query returns the current value of the maximum pulse amplitude.

Returned Format

```
[MTESt:HAMPlitude] <upper_limit><NL>
```

`<upper_limit>` A real number that represents the maximum amplitude in volts of a pulse as allowed by the pulse standard.

**Example**

This example returns the current upper pulse limit and prints it to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF"  !Response headers off
20  OUTPUT 707;"MTEST:HAMPLITUDE?"
30  ENTER 707;ULimit
40  PRINT ULimit
50  END
```

# IMPedance

Command

```
:MTESt:IMPedance {NONE | IMP75 | IMP100 | IMP110 |
IMP120}
```

The :MTESt:IMPedance command sets the desired probe impedance of the channel being used for mask testing. This impedance value is used when starting a mask test to determine whether or not the correct Infiniium probe is connected and in the case of the E2621A if the switch is set to the correct impedance value.

Infiniium has an AutoProbe interface that detects probes that have Probe ID resistors. If one of these probes is connected to the channel being mask tested and is not the correct probe for the selected impedance, a warning dialog box appears when the mask test is started from the human interface.

This command is meant to be used in the setup section of a mask file.

`NONE` Disables the probe impedance check.

`IMP75` Enables the probe impedance check for the E2622A probe.

`IMP100` Enables the probe impedance check for the E2621A probe with the switch set to the 100 ohm position.

`IMP110` Enables the probe impedance check for the E2621A probe with the switch set to the 110 ohm position.

`IMP120` Enables the probe impedance check for the E2621A probe with the switch set to the 120 ohm position.

**Example**

This example sets the probe impedance of the channel being used for mask testing to 100 ohms.

```
10 OUTPUT 707;"MTEST:IMPEDANCE IMP100"
20 END
```

Query

```
:MTESt:IMPedance?
```

The :MTESt:IMPedance? query returns the current value of the mask test impedance.

Returned Format

```
[:MTESt:IMPedance] {NONE | IMP75 | IMP100 | IMP110
| IMP120}<NL>
```

**Example**

This example returns the current value of the mask test impedance and prints the result to the computer screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"  !Response headers off
20  OUTPUT 707;":MTEST:IMPEDANCE?"
30  ENTER 707;Impedance
40  PRINT Impedance
50  END
```

# INVert

Command

```
:MTESt:INVert {{ON|1} | {OFF|0}}
```

The :MTESt:INVert command inverts the mask for testing negative-going pulses. The trigger level and mask offset are also adjusted. Not all masks support negative-going pulse testing, and for these masks, the command is ignored.

**Example**

This example inverts the mask for testing negative-going pulses.

```
10  OUTPUT 707;"MTEST:INVERT ON"
20  END
```

Query

```
:MTESt:INVert?
```

The :MTESt:INVert? query returns the current inversion setting.

Returned Format

```
[:MTESt:INVert] {1|0}<NL>
```

## LAMPlitude

Command

```
:MTESt:LAMPlitude <lower_limit>
```

The :MTESt:LAMPlitude command sets the minimum pulse amplitude value that passes the pulse standard. For some of the pulse communications standards, a pulse has a range of amplitude values and still passes the standard. This command sets the lower limit used during mask testing.

`<lower_limit>` A real number that represents the minimum amplitude in volts of a pulse as allowed by the pulse standard.

**Example**

This example sets the minimum pulse amplitude to 2.4 volts.

```
10  OUTPUT 707;"MTEST:LAMPLITUDE 2.4"
20  END
```

Query

```
:MTESt:LAMPlitude?
```

The :MTESt LAMPlitude? query returns the current value of the minimum pulse amplitude.

Returned Format

```
[:MTESt:LAMPlitude] <lower_limit><NL>
```

`<lower_limit>` A real number that represents the minimum amplitude in volts of a pulse as allowed by the pulse standard.

**Example**

This example returns the current lower pulse limit and prints it to the computer's screen.

```
10  OUTPUT 707;"SYSTEM:HEADER OFF  !Response headers off
20  OUTPUT 707;"MTEST:LAMPLITUDE?"
30  ENTER 707;ULimit
40  PRINT ULimit
50  END
```

# LOAD

Command

```
:MTESt:LOAD "<filename>"
```

The :MTESt:LOAD command loads the specified mask file. The default path for mask files is c:\scope\masks. To use a different path, specify the complete path and file name.

`<filename>` An MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used).

**Example**

This example loads the mask file named "140md_itu_1.msk".

```
10  OUTPUT 707;"MTEST:LOAD""c:\scope\masks\140md_itu_1.msk"""
20  END
```

## NREGions?

Query

```
:MTESt:NREGions?
```

The :MTESt:NREGions? query returns the number of regions that define the mask.

Returned Format

```
[:MTESt:NREGions] <regions><NL>
```

`<regions>` An integer from 0 to 8.

**Example**

This example returns the number of mask regions.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"
20  OUTPUT 707;":MTEST:NREGIONS?"
30  ENTER 707;Regions
40  PRINT Regions
50  END
```

# PROBe:IMPedance?

Query

```
:MTESt:PROBe:IMPedance?
```

The :MTESt:PROBe:IMPedance? query returns the impedance setting for the E2621A and E2622A probes for the current mask test channel.

Returned Format

```
[:MTESt:PROBe:IMPedance] <impedance><NL>
```

`<impedance>` An unquoted string: 75, 100, 110, 120, or NONE

**Example**

This example returns the impedance setting for the probe.

```
10  DIM Impedance$[20]
20  OUTPUT 707;":SYSTEM:HEADER OFF"
30  OUTPUT 707;":MTEST:PROBE:IMPEDANCE?"
40  ENTER 707;Impedance$
50  PRINT Impedance$
60  END
```

# RUMode

Command

```
:MTESt:RUMode {FORever | TIME, <time> | WAVeforms,
<number_of_waveforms>}
```

The :MTESt:RUMode command determines the termination conditions for the mask test. The choices are FORever, TIME, or WAVeforms.

If WAVeforms is selected, a second parameter is required indicating the number of failures that can occur or the number of samples or waveforms that are to be acquired.

`FORever` FORever runs the Mask Test until the test is turned off. This is used when you want a measurement to run continually and not to stop after a fixed number of failures. For example, you may want the Mask Test to run overnight and not be limited by a number of failures.

`TIME` TIME sets the amount of time in minutes that a mask test will run before it terminates.

`<time>` A real number: 0.1 to 1440.0

`WAVeforms` WAVeforms sets the maximum number of waveforms that are required before the mask test terminates.

`<number_of_waveforms>` An integer: 1 to 1,000,000,000.

**Example**

This example sets the mask test subsystem run until mode to continue testing until 500,000 waveforms have been gathered.

```
10 OUTPUT 707;"MTEST:RUMODE WAVEFORMS,500E3"
20 END
```

Query

```
:MTESt:RUMode?
```

The query returns the currently selected termination condition and value.

Returned Format

```
[:MTESt:RUMode] {FOREver | TIME,<time> | WAVeforms,
<number_of_waveforms>}<NL>
```

**Example**

This example gets the current setting of the mask test run until mode from the oscilloscope and prints it on the computer screen.

```
10 DIM MTEST_Runmode$[50]
20 OUTPUT 707; "MTEST:RUMODE?"
30 ENTER 707;":MTEST_Runmode$
40 PRINT MTEST_Runmode$
50 END
```

## RUMode:SOFailure

Command

```
:MTESt:RUMode:SOFailure {{ON|1} | {OFF|0}}
```

The :MTESt:RUMode:SOFailure command enables or disables the Stop On Failure run until criteria. When a mask test is run and a mask violation is detected, the mask test is stopped and the acquisition system is stopped.

**Example**

This example enables the Stop On Failure run until criteria.

```
10  OUTPUT 707;":MTEST:RUMODE:SOFAILURE ON"
20  END
```

Query

```
:MTESt:SOFailure?
```

The :MTESt:SOFailure? query returns the current state of the Stop on Failure control.

Returned Format

```
[:MTESt:SOFailure] {1|0}<NL>
```

# SCALe:BIND

Command

```
:MTESt:SCALe:BIND {{ON|1} | {OFF|0}}
```

The :MTESt:SCALe:BIND command enables or disables Bind 1 & 0 Levels (Bind -1 & 0 Levels for inverted masks) control. If the Bind 1 & 0 Levels control is enabled, the 1 Level and the 0 Level controls track each other. Adjusting either the 1 Level or the 0 Level control shifts the position of the mask up or down without changing its size. If the Bind 1 & 0 Levels control is disabled, adjusting either the 1 Level or the 0 Level control changes the vertical height of the mask.

If the Bind -1 & 0 Levels control is enabled, the -1 Level and the 0 Level controls track each other. Adjusting either the -1 Level or the 0 Level control shifts the position of the mask up or down without changing its size. If the Bind -1 & 0 Levels control is disabled, adjusting either the -1 Level or the 0 Level control changes the vertical height of the mask.

**Example**

This example enables the Bind 1 & 0 Levels control.

```
10 OUTPUT 707;"MTEST:BIND ON"
20 END
```

Query

```
:MTESt:SCALe:BIND?
```

The :MTESt:SCALe:BIND? query returns the value of the Bind 1&0 control (Bind -1&0 for inverted masks).

Returned Format

```
[:MTESt:SCALe:BIND?] {1|0}<NL>
```

# SCALe:X1

Command

```
:MTESt:SCALe:X1 <x1_value>
```

The :MTESt:SCALe:X1 command defines where X=0 in the base coordinate system used for mask testing. The other X-coordinate is defined by the SCALe:XDELta command. Once the X1 and XDELta coordinates are set, all X values of vertices in the mask regions are defined with respect to this value, according to the equation: $X = (X \times \Delta X) + X1$

Thus, if you set X1 to 100 ms, and XDELta to 100 ms, an X value of 0.100 is a vertex at 110 ms.

The oscilloscope uses this equation to normalize vertices. This simplifies reprogramming to handle different data rates. For example, if you halve the period of the waveform of interest, you need only to adjust the XDELta value to set up the mask for the new waveform.

`<x1_value>` A time value specifying the location of the X1 coordinate, which will then be treated as X=0 for mask regions coordinates.

**Example**

This example sets the X1 coordinate at 150 ms.

```
10 OUTPUT 707;":MTEST:SCALE:X1 150E-3"
20 END
```

Query

```
:MTESt:SCALe:X1?
```

The :MTESt:SCALe:X1? query returns the current X1 coordinate setting.

Returned Format

```
[:MTESt:SCALe:X1] <x1_value><NL>
```

**Example**

This example gets the current setting of the X1 coordinate from the oscilloscope and prints it on the computer screen.

```
10 DIM Scale_x1$[50]
20 OUTPUT 707;":MTEST:SCALE:X1?"
30 ENTER 707;Scale_x1$
40 PRINT Scale_x1$
50 END
```

# SCALe:XDELta

Command

```
:MTESt:SCALe:XDELta <xdelta_value>
```

The :MTESt:SCALe:XDELta command defines the position of the X2 marker with respect to the X1 marker. In the mask test coordinate system, the X1 marker defines where X=0; thus, the X2 marker defines where X=1.

Because all X vertices of the regions defined for mask testing are normalized with respect to X1 and ΔX, redefining ΔX also moves those vertices to stay in the same locations with respect to X1 and ΔX. Thus, in many applications, it is best if you define XDELta as a pulse width or bit period. Then a change in data rate without corresponding changes in the waveform can easily be handled by changing ΔX.

The X-coordinate of polygon vertices is normalized using this equation:

$$X = (X \times \Delta X) + X1$$

`<xdelta_value>` A time value specifying the distance of the X2 marker with respect to the X1 marker.

**Example**

Assume that the period of the waveform you wish to test is 1 ms. Then the following example will set ΔX to 1 ms, ensuring that the waveform's period is between the X1 and X2 markers.

```
10 OUTPUT 707;":MTEST:SCALE:XDELTA 1E-6:
20 END
```

Query

```
:MTESt:SCALe:XDELta?
```

The :MTESt:SCALe:XDELta? query returns the current value of ΔX.

Returned Format

```
[:MTESt:SCALe:XDELta] <xdelta_value><NL>
```

**Example**

This example gets the value of ΔX from the oscilloscope and prints it on the computer screen.

```
10 DIM Scale_xdelta$[50]
20 OUTPUT 707;":MTEST:SCALE:XDELTA?"
30 ENTER 707;Scale_xdelta$
40 PRINT Scale_xdelta$
50 END
```

## SCALe:Y1

**Command**

```
:MTESt:SCALe:Y1 <y_value>
```

The :MTESt:SCALe:Y1 command defines where Y=0 in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries set by SCALe:Y1 and SCALe:Y2 according to the equation: $Y = (Y \times (Y2 - Y1)) + Y1$

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of 0.100 in a vertex is at 190 mV.

`<y1_value>` A voltage value specifying the point at which Y=0.

**Example**

This example sets the Y1 marker to -150 mV.

```
10 OUTPUT 707; ":MTEST:SCALE:Y1 -150E-3"
20 END
```

**Query**

```
:MTESt:SCALe:Y1?
```

The SCALe:Y1? query returns the current setting of the Y1 marker.

**Returned Format**

```
[:MTESt:SCALe:Y1] <y1_value><NL>
```

**Example**

This example gets the setting of the Y1 marker from the oscilloscope and prints it on the computer screen.

```
10 DIM Scale_y1$[50]
20 OUTPUT 707;":MTEST:SCALE:Y1?"
30 ENTER 707;Scale_y1$
40 PRINT Scale_y1$
50 END
```

# SCALe:Y2

Command

```
:MTESt:SCALe:Y2 <y2_value>
```

The :MTESt:SCALe:Y2 command defines the Y2 marker in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries defined by SCALe:Y1 and SCALe:Y2 according to the following equation: $Y = (Y \times (Y2 - Y1)) + Y1$

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of 0.100 in a vertex is at 190 mV.

`<y2_value>` A voltage value specifying the location of the Y2 marker.

**Example**

This example sets the Y2 marker to 2.5 V.

```
10 OUTPUT 707;":MTEST:SCALE:Y2 2.5"
20 END
```

Query

```
:MTESt:SCALe:Y2?
```

The SCALe:Y2? query returns the current setting of the Y2 marker.

Returned Format

```
[:MTESt:SCALe:Y2] <y2_value><NL>
```

**Example**

This example gets the setting of the Y2 marker from the oscilloscope and prints it on the computer screen.

```
10 DIM Scale_y2$[50]
20 OUTPUT 707;":MTEST:SCALE:Y2?"
30 ENTER 707;Scale_y2$
40 PRINT Scale_y2$
50 END
```

## SOURce

Command

```
:MTESt:SOURce {CHANnel<N> | FUNCtion<M>}
```

The :MTESt:SOURce command selects the channel which is configured by the commands contained in a mask file when it is loaded.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

<M> An integer, 1 - 4.

**Example**

This example selects channel 1 as the mask test source.

```
10 OUTPUT 707; "MTEST:SOURCE CHANNEL1"
20 END
```

Query

```
:MTESt:SOURce?
```

The :MTESt:SOURce? query returns the channel which is configured by the commands contained in the current mask file.

Returned Format

```
[:MTESt:SOURce] {CHANnel<N> | FUNCtion<M>}<NL>
```

**Example**

This example gets the mask test source setting and prints the result on the computer display.

```
10 DIM Amask_source$[30]
20 OUTPUT 707;"MTEST:SOURCE?"
30 ENTER 707;Amask_source$
40 PRINT Amask_source$
50 END
```

# STARt | STOP

Command

```
:MTESt:STARt|STOP
```

The :MTESt:STARt|STOP command starts or stops the mask test. The :MTESt:STARt command also starts the oscilloscope acquisition system. The :MTESt:STOP command does not stop the acquisition system.

**Example**

This example starts the mask test and acquisition system.

```
10  OUTPUT 707;"MTEST:START"
20  END
```

# STIMe

Command

```
:MTESt:STIMe <timeout>
```

The :MTESt:STIMe command sets the timeout value for the Autoalign feature. If the oscilloscope is unable to align the mask to your waveform within the specified timeout value, it will stop trying to align and will report an alignment failure.

`<timeout>` An integer from 1 to 120 seconds representing the time between triggers (not the time that it takes to finish the alignment.)

**Example**

This example sets the timeout value for the Autoalign feature to 10 seconds.

```
10 OUTPUT 707;"MTEST:STIMe 10"
20 END
```

Query

```
:MTESt:STIMe?
```

The query returns timeout value for the Autoalign feature.

Returned Format

```
[:MTESt:STIMe] <timeout><NL>
```

**Example**

This example gets the timeout setting and prints the result on the computer display.

```
10 OUTPUT 707;"MTEST:STIME?"
30 ENTER 707;Value
40 PRINT Value
50 END
```

# TITLe?

**Query**

```
:MTESt:TITLe?
```

The :MTESt:TITLe? query returns the mask title which is a string of up to 23 characters. The title is displayed in the mask test dialog box and mask test tab when a mask file is loaded.

**Returned Format**

```
[:MTESt:TITLe] <mask_title><NL>
```

`<mask_title>` A string of up to 23 ASCII characters which is the mask title.

**Example**

This example places the mask title in the string variable and prints the contents to the computer's screen.

```
10  DIM Title$[24]
20  OUTPUT 707;":MTEST:TITLE?"
30  ENTER 707;Title$
40  PRINT Title$
50  END
```

# TRIGger:SOURce

Command

```
:MTESt:TRIGger:SOURce {CHANnel<N> | EXTernal}
```

The :MTESt:TRIGger:SOURce command sets the channel or function to use as the trigger. The EXTernal parameter is only available on the 54810A and 54820A oscilloscopes. Mask testing must be enabled before using this command.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**Example**

This example sets the mask trigger source to channel 1.

```
10 OUTPUT 707;"MTEST:TRIGGER:SOURCE CHANNEL1"
20 END
```

Query

```
:MTESt:TRIGger:SOURce?
```

The query returns the currenly selected mask test trigger source.

Returned Format

```
[:MTESt:TRIGger] {CHANnel<N> | EXTernal}<NL>
```

**Example**

This example gets the trigger source setting and prints the result on the computer display.

```
10 DIM Amask_source$[30]
20 OUTPUT 707;"MTEST:TRIGGER:SOURCE?"
30 ENTER 707;Amask_source$
40 PRINT Amask_source$
50 END
```

22

# Self-Test Commands

# Self-Test Commands

The SELFtest subsystem commands set up the self-test dialog and run the Infiniium-Series Oscilloscopes Self-Tests.

**Enclose File Name in Quotation Marks**

When specifying a file name, you must enclose it in quotation marks.

These SELFtest commands and queries are implemented in the Infiniium Oscilloscopes:

- AttenSET?
- CANCel
- SCOPETEST

# AttenSET?

Query

```
:SELFtest:AttenSET? [<atten_set_chan>]
```

The :SELFtest:AttenSET? query returns the channel number and number of relay actuations for each channel.

Returned Format

```
[SELFtest:AttenSET]
Channel<space><channel_number>,<num_actuations>}{[,Channel
<space><channel_number>,<num_actuations>]...}<NL>
```

<atten_set_chan> {CHAN<N> | EXTernal | ALL}

<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

ALL is the default, and performs the same function as if no parameter is specified. External is only available on 2-channel oscilloscopes, which include 54810/54820.

**Example**

This example queries the oscilloscope for channels and the number of relay actuations for each channel.

```
10 DIM Txt$[85]
20 OUTPUT 707;":SELF:ASET?"
30 ENTER 707;Txt$
40 PRINT Txt$
50 END
```

## CANCel

Command

```
:SELFtest:CANCel
```

The :SELFtest:CANCel command stops the currenly running selftest.

**Example**

This example stops the currently running selftest.

```
10 OUTPUT 707;":SELF:CANC"
20 END
```

# SCOPETEST

Command

```
:SELFtest:SCOPETEST
```

The :SELFtest:SCOPETEST command brings up the self-test dialog in customer self-test mode (Service Extensions Off) and runs the test, "Scope Self Tests." Use the :SELFtest:SCOPETEST? query to determine the status of the test.

**Example**

This example brings up the self-test dialog and runs the oscilloscope self-tests.

```
10 OUTPUT 707;":SELF:SCOPETEST"
20 END
```

Query

```
:SELFtest:SCOPETEST?
```

Returned Format

```
[:SELFtest:SCOPETEST] <test_name>,<test_status>,
<time_stamp><NL>
```

| <test_status> | **Status Description** |
|---|---|
| FAILED | Test completed and failed. |
| PASSED | Test completed and passed. |
| WARNING | Test passed but warning message was issued. |
| CANCELLED | Test was cancelled by user. |
| NODATA | Self-tests have not been executed on this instrument. |
| INPROGRESS | Test is in progress. |

`<test_name>` A string as follows: "Scope Self Tests".

`<time_stamp>` The time stamp follows the test name and test status, and is the part of the returned string that includes the date and time, in the format: "29 AUG 1997 10:13:35".

**Example**

This example places the current status of the self-test in the string variable, Txt$, then prints the contents of the variable to the computer's screen.

```
10 DIM Txt$[64]
20 OUTPUT 707;":SELF:SCOPETEST?"
30 ENTER 707;Txt$
40 PRINT Txt$
50 END
```

23

# Time Base Commands

# Time Base Commands

The TIMebase subsystem commands control the horizontal (X axis) oscilloscope functions. These TIMebase commands and queries are implemented in the Infiniium Oscilloscopes:

- DELay
- POSition
- RANGe
- REFerence
- SCALe
- VIEW
- WINDow:DELay
- WINDow:POSition
- WINDow:RANGe
- WINDow:SCALe

# DELay

**Command**

```
:TIMebase:DELay <delay_value>
```

The :TIMebase:DELay command sets the time interval between the trigger event and the delay reference point. The delay reference point is set with the :TIMebase:REFerence command.

**This Command is Provided for Compatibility**

This command has the same function as the :TIMebase:POSition command, and is provided for compatibility with programs written for previous oscilloscopes. The preferred command for compatibility with Infiniium Oscilloscopes is :TIMebase:POSition.

`<delay_value>` A real number for the time in seconds from trigger to the delay reference point. The maximum value depends on the time/division setting.

**Example**

This example sets the delay to 2 ms.

```
10  OUTPUT 707;":TIMEBASE:DELAY 2E-3"
20  END
```

Query

```
:TIMebase:DELay?
```

The :TIMebase:DELay? query returns the current delay value in seconds.

Returned Format

```
[:TIMebase:DELay] <delay_value><NL>
```

**Example**

This example places the current delay value in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:DELAY?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

**Turn Off Headers**

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

**See Also**

The :TIMebase:POSition command performs the same function as this command and is preferred for new programs.

# POSition

Command

```
:TIMebase:POSition <position_value>
```

The :TIMebase:POSition command sets the time interval between the trigger event and the delay reference point. The delay reference point is set with the :TIMebase:REFerence command.

`<position_value>` A real number for the time in seconds from trigger to the delay reference point. The maximum value depends on the time/division setting.

**Example**

This example sets the delay position to 2 ms.

```
10  OUTPUT 707;":TIMEBASE:POSITION 2E-3"
20  END
```

Query

```
:TIMebase:POSition?
```

The :TIMebase:POSition? query returns the current delay value in seconds.

Returned Format

```
[:TIMebase:POSition] <position_value><NL>
```

**Example**

This example places the current delay value in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:POSITION?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# RANGe

Command

```
:TIMebase:RANGe <full_scale_range>
```

The :TIMebase:RANGe command sets the full-scale horizontal time in seconds. The range value is ten times the time-per-division value.

`<full_scale_range>` A real number for the horizontal time, in seconds.
The 54845A and 54835A have 1 ns (100 ps/div) to 50 s (5 s/div).
The 54810/20/15/25/A models have 5 ns (500 ps/div) to 50 s (5 s/div).

**Example**

This example sets the full-scale horizontal range to 10 ms.

```
10  OUTPUT 707;":TIMEBASE:RANGE 10E-3"
20  END
```

Query

```
:TIMebase:RANGe?
```

The :TIMebase:RANGe? query returns the current full-scale horizontal time.

Returned Format

```
[:TIMebase:RANGe] <full_scale_range><NL>
```

**Example**

This example places the current full-scale horizontal range value in the numeric variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:RANGE?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

## REFerence

**Command**

```
:TIMebase:REFerence {LEFT | CENTer | RIGHt}
```

The :TIMebase:REFerence command sets the delay reference to the left, center, or right side of the display.

**Example**

This example sets the delay reference to the center of the display.

```
10  OUTPUT 707;":TIMEBASE:REFERENCE CENTER"
20  END
```

**Query**

```
:TIMebase:REFerence?
```

The :TIMebase:REFerence? query returns the current delay reference position.

**Returned Format**

```
[:TIMebase:REFerence] {LEFT | CENTer | RIGHt}<NL>
```

**Example**

This example places the current delay reference position in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":TIMEBASE:REFERENCE?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# SCALe

Command

```
:TIMebase:SCALe <time>
```

The :TIMebase:SCALe command sets the time base scale. This corresponds to the horizontal scale value displayed as time/div on the oscilloscope screen.

`<time>` A real number for the time value, in seconds per division.

**Example**

This example sets the scale to 10 ms/div.

```
10  OUTPUT 707;":TIMEBASE:SCALE 10E-3"
20  END
```

Query

```
:TIMebase:SCALe?
```

The :TIMebase:SCALe? query returns the current scale time setting.

Returned Format

```
[:TIMebase:SCALe] <time><NL>
```

**Example**

This example places the current scale value in the numeric variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:SCALE?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

# VIEW

Command

```
:TIMebase:VIEW {MAIN | WINDow}
```

The :TIMebase:VIEW command turns the delayed displayed view on and off. This is the same as using the front panel Delayed button.

**Example**

This example turns the delayed view on.

```
10  OUTPUT 707;":TIMEBASE:VIEW WINDOW"
20  END
```

Query

```
:TIMebase:VIEW?
```

The :TIMebase:VIEW? query returns Infiniium's current view.

Returned Format

```
[:TIMebase:VIEW] {MAIN | WINDow}<NL>
```

**Example**

This example places the current view in the string variable, State$, then prints the contents of the variable to the computer's screen.

```
10  DIM State$[50]!Dimension variable
20  OUTPUT 707;":TIMEBASE:VIEW?"
30  ENTER 707;State$
40  PRINT State$
50  END
```

# WINDow:DELay

Command

```
:TIMebase:WINDow:DELay <delay_value>
```

The :TIMebase:WINDow:DELay sets the horizontal position in the delayed view of the main sweep. The range for this command is determined by the main sweep range and the main sweep horizontal position. The value for this command must keep the time base window within the main sweep range.

**This Command is Provided for Compatibility**

This command has the same function as the :TIMebase:WINDow:POSition command, and is provided for compatibility with programs written for previous oscilloscopes. The preferred command for compatibility with Infiniium Oscilloscopes is :TIMebase:WINDow:POSition.

`<delay_value>` A real number for the time in seconds from the trigger event to the delay reference point. The maximum position depends on the main sweep range and the main sweep horizontal position.

**Example**

This example sets the time base window delay position to 20 ns.

```
10  OUTPUT 707;":TIMEBASE:WINDOW:DELAY 20E-9"
20  END
```

Query

```
:TIMebase:WINDow:DELay?
```

The :TIMebase:WINDow:DELay? query returns the current horizontal position in the delayed view.

Returned Format

```
[:TIMebase:WINDow:DELay] <delay_position><NL>
```

**Example**

This example places the current horizontal position in the delayed view in the numeric variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:WINDOW:DELAY?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

**See Also**

The :TIMebase:WINDow:POSition command performs the same function as this command and should be used in new programs.

# WINDow:POSition

Command

```
:TIMebase:WINDow:POSition <position_value>
```

The :TIMebase:WINDow:POSition sets the horizontal position in the delayed view of the main sweep. The range for this command is determined by the main sweep range and the main sweep horizontal position. The value for this command must keep the time base window within the main sweep range.

<position_value> A real number for the time in seconds from the trigger event to the delay reference point. The maximum position depends on the main sweep range and the main sweep horizontal position.

**Example**

This example sets the time base window delay position to 20 ns.

```
10  OUTPUT 707;":TIMEBASE:WINDOW:POSITION 20E-9"
20  END
```

Query

```
:TIMebase:WINDow:POSition?
```

The :TIMebase:WINDow:POSition? query returns the current horizontal position in the delayed view.

Returned Format

```
[:TIMebase:WINDow:POSition] <position_value><NL>
```

**Example**

This example places the current horizontal position in the delayed view in the numeric variable, Setting, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:WINDOW:POSITION?"
30  ENTER 707;Setting
40  PRINT Setting
50  END
```

## WINDow:RANGe

**Command**

```
:TIMebase:WINDow:RANGe <full_scale_range>
```

The :TIMebase:WINDow:RANGe command sets the full-scale range of the delayed view. The range value is ten times the time per division of the delayed view. The maximum range of the delayed view is the current main range. The minimum delayed view range is 10 ps (1 ps/div).

`<full_scale_range>` A real number for the full-scale range of the time base window, in seconds.

**Example**

This example sets the full-scale range of the delayed view to 100 ns.

```
10  OUTPUT 707;":TIMEBASE:WINDOW:RANGE 100E-9"
20  END
```

**Query**

```
:TIMebase:WINDow:RANGe?
```

The :TIMebase:WINDow:RANGe? query returns the current full-scale range of the delayed view.

**Returned Format**

```
[:TIMebase:WINDow:RANGe] <full_scale_range><NL>
```

**Example**

This example reads the current full-scale range of the delayed view into the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":TIMEBASE:WINDOW:RANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

# WINDow:SCALe

Command

```
:TIMebase:WINDow:SCALe <time>
```

The :TIMebase:WINDow:SCALe command sets the time/div in the delayed view. This command rescales the horizontal components of displayed waveforms.

`<time>` A real number for the time applied to scale the waveforms, ranging from 20 µs to 2.00E-5.

**Example**

This example sets the scale of the time base window to 2 milliseconds/div.

```
10  OUTPUT 707;":TIMEBASE:WINDOW:SCALE 2E-3"
20  END
```

Query

```
:TIMebase:WINDow:SCALe?
```

The :TIMebase:WINDow:SCALe? query returns the scaled window time, in seconds/div.

Returned Format

```
[:TIMebase:WINDow:SCALe] <time><NL>
```

# Trigger Commands

# Trigger Commands

The oscilloscope trigger circuitry helps you locate the waveform you want to view. There are several different types of triggering, but the one that is used most often is edge triggering. Edge triggering identifies a trigger condition by looking for the slope (rising or falling) and voltage level (trigger level) on the source you select. Any input channel, auxiliary input trigger (only in 4-channel oscilloscopes), line, or external trigger (only in 2-channel oscilloscopes) can be used as the trigger source.

The commands in the TRIGger subsystem define the conditions for triggering. Many of the commands in the TRIGger subsystem are used in more than one of the trigger modes. The command set has been defined to closely represent the front-panel trigger menus. As a trade-off, there may be less compatibility between Infiniium Oscilloscopes and command sets for previous oscilloscopes. Infiniium Oscilloscopes still accept some commands for compatibility with previous instruments. An alternative command that is accepted by the oscilloscope is noted for a particular command.

These TRIGger commands and queries are implemented in the Infiniium Oscilloscopes:

- HOLDoff
- HYSTeresis
- LEVel
- SWEep

- MODE {EDGE | GLITch | ADVanced}

- EDGe {:SLOPe | :SOURce | :COUPling}
- GLITch {:POLarity | :SOURce | :WIDTh}
- :ADVanced:MODE {COMM | DELay | PATTern | STATe | TV | VIOLation}
- :ADVanced:MODE COMM
- :ADVanced:COMM:{BWIDth | ENCode | LEVel | PATTern | POLarity | SOURce}

- :ADVanced:MODE DELay
- :ADVanced:DELay
- :ADVanced:DELay:MODE {EDLY | TDLY}

- :ADVanced:MODE PATTern
- :ADVanced:PATTern {:CONDition | :LOGic}

- :ADVanced:MODE STATe
- :ADVanced:STATE {:CLOCk | :CONDition | :LOGic | :SLOPe}

- :ADVanced:MODE TV
- :ADVanced:TV
- :ADVanced:TV:MODE {L525 | L625 | L875 | UDTV}

- :ADVanced:MODE VIOLation
- :ADVanced:VIOLation (See the following list.)

The :TRIGger:ADVanced:VIOLation modes and commands described in this chapter include:

- :VIOLation:MODE SETup

- :VIOLation:SETup:MODE SETup
- :VIOLation:SETup:SETup:CSOurce
- :VIOLation:SETup:SETup:CSOurce:LEVel
- :VIOLation:SETup:SETup:CSOurce:EDGE
- :VIOLation:SETup:SETup:DSOurce
- :VIOLation:SETup:SETup:DSOurce:LTHReshold
- :VIOLation:SETup:SETup:DSOurce:HTHReshold
- :VIOLation:SETup:SETup:TIME
- :VIOLation:SETup:MODE HOLD
- :VIOLation:SETup:HOLD:CSOurce
- :VIOLation:SETup:HOLD:CSOurce:LEVel
- :VIOLation:SETup:HOLD:CSOurce:EDGE
- :VIOLation:SETup:HOLD:DSOurce
- :VIOLation:SETup:HOLD:DSOurce:LTHReshold
- :VIOLation:SETup:HOLD:DSOurce:HTHReshold
- :VIOLation:SETup:HOLD:TIME

- :VIOLation:SETup:MODE SHOLd
- :VIOLation:SETup:SHOLd:CSOurce
- :VIOLation:SETup:SHOLd:CSOurce:LEVel
- :VIOLation:SETup:SHOLd:CSOurce:EDGE
- :VIOLation:SETup:SHOLd:DSOurce
- :VIOLation:SETup:SHOLd:DSOurce:LTHReshold
- :VIOLation:SETup:SHOLd:DSOurce:HTHReshold
- :VIOLation:SETup:SHOLd:SetupTIMe
- :VIOLation:SETup:SHOLd:HoldTIMe

- :VIOLation:MODE TRANsition
- :VIOLation:TRANsition:SOURce
- :VIOLation:TRANsition:TYPE
- :VIOLation:TRANsition:GTHan
- :VIOLation:TRANsition:LTHan

- :VIOLation:MODE PWIDth
- :VIOLation:PWIDth:SOURce
- :VIOLation:PWIDth:POLarity
- :VIOLation:PWIDth:DIRection
- :VIOLation:PWIDth:WIDTh

# Organization of Trigger Modes and Commands

The trigger modes are summarized in the next section. In addition, each mode is described before its set of commands in the following sections.

These general trigger commands are described first.

- HOLDoff
- HYSTeresis
- LEVel
- SWEep

The following sections in this chapter describe the individual trigger modes and commands, and are organized in this order:

- EDGE
- GLITch
- ADVanced
  - COMM
  - DELay
  - PATTern
  - STATe
  - TV
  - VIOLation

Make sure the oscilloscope is in the proper trigger mode for the command you want to send. One method of ensuring that the oscilloscope is in the proper trigger mode is to send the :TRIGger:MODE command in the same program message as the parameter to be set.

For example, these commands place the instrument in the advanced triggering mode you select:

```
:TRIGger:MODE ADVanced
:TRIGger:ADVanced:MODE <Advanced_trigger_mode>
```

`<Advanced_trigger_mode>` Advanced trigger modes include COMM, DELay, PATTern, STATe, TV, and VIOLation. Each mode is described with its command set in this chapter.

**Summary of Trigger Commands**

The following table lists the TRIGger subsystem commands that are available for each trigger mode.

**Table 24-1**

**Valid Commands for Specific Trigger Modes**

| Main Level | EDGE | GLITCH |
|---|---|---|
| HOLDoff | COUPling | POLarity |
| HYSTeresis | SLOPe | SOURce |
| LEVel | SOURce | WIDTh |
| SWEep | | |

**ADVANCED TRIGGERING MODES AND COMMANDS**

| COMM | DELay | PATTern | STATe | TV | VIOLation |
|---|---|---|---|---|---|
| BWIDth | MODE | CONDition | CLOCk | MODE | MODE |
| ENCode | EDLY | LOGic | CONDition | {L525 \| L625 \| L875 \| UDTV} | PWIDth |
| LEVel | ARM | | LOGic | STV | SETup |
| PATTern | EVENt | | SLOPe | FIELd | TRANsition |
| POLarity | TRIGger | | | LINE | |
| SOURce | TDLY | | | SOURce | (See the :TRIGger:ADVanced:VIOLation commands in this chapter for descriptions of the various violation modes and commands.) |
| | ARM | | | SPOLarity | |
| | DELay | | | UDTV | |
| | TRIGger | | | ENUMber | |
| | | | | PGTHan | |
| | | | | PLTHan | |
| | | | | POLarity | |
| | | | | SOURce | |
| | | | | EDGE | |

**Use :TRIGger:SWEep to Select Sweep Mode**

Select the Infiniium Oscilloscope's Auto, Triggered, or Single Sweep mode with :TRIGger:SWEep {AUTO | TRIGgered | SINGle}.

# Trigger Modes

Command

```
:TRIGger:MODE {EDGE | GLITch | ADVanced}
```

The :TRIGger:MODE command selects the trigger mode.

**Table 24-2** **Trigger Mode Settings**

| Mode | Definition |
|---|---|
| EDGE | Edge trigger mode. |
| GLITch | Trigger on a pulse that has a width less than a specified amount of time. |
| ADVanced | Allows access to the DELay, PATTern, STATe, TV, and VIOLation modes. |
| COMM | COMM mode lets you trigger on a serial pattern of bits in a waveform. |
| DELay | Delay by Events mode lets you view pulses in your waveform that occur a number of events after a specified waveform edge. Delay by Time mode lets you view pulses in your waveform that occur a long time after a specified waveform edge. |
| PATTern | Pattern triggering lets you trigger the oscilloscope using more than one channel as the trigger source. You can also use pattern triggering to trigger on a pulse of a given width. |
| STATe | State triggering lets you set the oscilloscope to use several channels as the trigger source, with one of the channels being used as a clock waveform. |
| TV | TV trigger mode lets you trigger the oscilloscope on one of the standard television waveforms. You can also use this mode to trigger on a custom television waveform that you define. |
| VIOLation | Trigger violation modes: Pulse WIDth, SETup, TRANsition. |

Query

```
:TRIGger:MODE?
```

The query returns the currently selected trigger mode.

Returned Format

```
[:TRIGger:MODE] {EDGE | GLITch | ADVanced}<NL>
```

# HOLDoff

**Command** `:TRIGger:HOLDoff <holdoff_time>`

The :TRIGger:HOLDoff command specifies the amount of time the oscilloscope should wait after receiving a trigger before enabling the trigger again.

`<holdoff_time>` A real number for the holdoff time, ranging from 60 ns to 320 ms.

**Query** `:TRIGger:HOLDoff?`

The query returns the current holdoff value for the current mode.

**Returned Format** `[:TRIGger:HOLDoff] <holdoff><NL>`

# HTHReshold

Command

```
:TRIGger:HTHReshold
{{CHANnel<N>|EXTernal},<level>}}
```

This command specifies the high threshold voltage level for the selected trigger source. Set the high threshold level to a value considered to be a high level for your logic family; your data book gives two values, $V_{IH}$ and $V_{OH}$.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the voltage level for the trigger source.

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

Query

```
:TRIGger:HTHReshold? {CHANnel<N>|EXTernal}
```

The query returns the currently defined high threshold voltage level for the trigger source.

Returned Format

```
[:TRIGger:HTHReshold {CHANnel<N>|EXTernal},] <level><NL>
```

## HYSTeresis

Command

```
:TRIGger:HYSTeresis {NORMal|NREJect}
```

The :TRIGger:HYSTeresis command specifies the trigger hysteresis (noise reject) as either normal or maximum. The NORMal option is the typical hysteresis selection. The NREJect (noise reject) option gives maximum hysteresis but the lowest trigger bandwidth.

**Trigger Hysteresis is Available on Most Infiniium Models**

The :TRIGger:HYSTeresis command is available on all Infiniium-Series Oscilloscopes except the 54845A and 54835A.

Query

```
:TRIGger:HYSTeresis?
```

The query returns the current hysteresis setting.

Returned Format

```
[:TRIGger:HYSTeresis] {NORMal|NREJect}<NL>
```

# LEVel

Command

```
:TRIGger:LEVel {{CHANnel<N>|AUX|EXTernal},<level>}}
```

The :TRIGger:LEVel command specifies the trigger level on the specified channel for the trigger source. Only one trigger level is stored in the oscilloscope for each channel. This level applies to the channel throughout the trigger dialogs (Edge, Glitch, and Advanced). This level also applies to all the High Threshold (HTHReshold) values in the Advanced Violation menus.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the trigger level on the specified channel, External Trigger, or Auxilliary Trigger Input.

Query

```
:TRIGger:LEVel? {CHANnel<N>|AUX|EXTernal}
```

The query returns the specified channel's trigger level.

Returned Format

```
[:TRIGger:LEVel {CHANnel<N>|AUX|EXTernal},] <level><NL>
```

**EXTernal and AUXiliary are Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

AUXiliary is only available in 4-channel Infiniium Oscilloscope models (including the 54815A, 54825A, 54835A, and 54845A).

## LTHReshold

Command

```
:TRIGger:LTHReshold
{{CHANnel<N>|EXTernal},<level>}}
```

This command specifies the low threshold voltage level for the selected trigger source. This command specifies the low threshold voltage level for the selected trigger source. Set the low threshold level to a value considered to be a low level for your logic family; your data book gives two values, $V_{IL}$ and $V_{OL}$.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the voltage level for the trigger source.

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

Query

```
:TRIGger:LTHReshold? {CHANnel<N>|EXTernal}
```

The query returns the currently defined low threshold for the trigger source.

Returned Format

```
[:TRIGger:LTHReshold {CHANnel<N>|EXTernal},] <level><NL>
```

## SWEep

Command

```
:TRIGger:SWEep {AUTO|TRIGgered|SINGle}
```

The :TRIGger:SWEep command selects the oscilloscope sweep mode.

`<AUTO>` When you select AUTO, if a trigger event does not occur within a time determined by the oscilloscope settings, the oscilloscope automatically forces a trigger which causes the oscilloscope to sweep. If the frequency of your waveform is 50 Hz or less, you should not use the AUTO sweep mode because it is possible that the oscilloscope will automatically trigger before your waveform trigger occurs.

`<TRIGgered>` When you select TRIGgered, if no trigger occurs, the oscilloscope will not sweep, and the previously acquired data will remain on the screen.

`<SINGle>` When you select SINGle, if no trigger occurs, the oscilloscope will not sweep, and the previously acquired data will remain on the screen.

Query

```
:TRIGger:SWEep?
```

The query returns the specified channel's trigger level.

Returned Format

```
[:TRIGger:SWEep] {AUTO|TRIGgered|SINGle}<NL>
```

# Edge Trigger Mode and Commands

The oscilloscope identifies an edge trigger by looking for the specified slope (rising edge or falling edge) of your waveform. Once the slope is found, the oscilloscope will trigger when your waveform crosses the trigger level.

The Edge Trigger Mode is the easiest trigger mode to understand and use from the front panel or over the remote interface, because it has the least number of parameters to be set. This explanation of the trigger mode commands follow the front-panel keys very closely. Refer to the *online help file* for further explanations of the trigger operation.

In the Edge Trigger Mode, you must set the trigger source using the :TRIGger:EDGE:SOURce command. This selects the source that the oscilloscope triggers on. The argument for the :TRIGger:EDGE:SOURce command is CHANnel<n> (where n = 1 through 4) AUX, or LINE (or External for 2-channel units).

After setting the trigger source, set the trigger slope. The actual edge that creates the trigger is set with the :TRIGger:EDGE:SLOPe command. You can set this command to POSitive or NEGative for each of the sources, except LINE.

Set the trigger level for the trigger source. Only one trigger level is stored in the oscilloscope for each channel. The trigger level values that are set in the Edge Trigger Mode are used for all modes. Any levels set in the PATTern, STATe, or DELay, TV, or violation (high threshold) modes set the levels for the EDGE mode. LINE has no level.

Available trigger conditioning includes HOLDoff, HYSTeresis (Noise Reject) and COUPling.

**Set the Mode Before Executing Commands**

Before you can execute the :TRIGger:EDGE commands, set the mode by entering:

```
:TRIGger:MODE EDGE
```

This command sets the conditions for the EDGE slope and source trigger commands.

To query the oscilloscope for the trigger mode, enter:

```
:TRIGger:MODE?
```

You set up the :TRIGger:EDGE commands with the following commands and queries:

- COUPling
- SLOPe
- SOURce

# EDGE:COUPling

Command

```
:TRIGger:EDGE:COUPling {AC|DC|LFReject|HFReject}
```

The :TRIGger:EDGE:COUPling command sets the trigger coupling when :TRIG:EDGE:SOURce is set to one of the channels, or to External (for 2-channel oscilloscope models).

Query

```
:TRIGger:EDGE:COUPling?
```

The query returns the currently selected coupling for the specified edge trigger source.

Returned Format

```
[:TRIGger:EDGE:COUPling] {AC|DC|LFReject|HFReject}<NL>
```

## EDGE:SLOPe

**Command**

```
:TRIGger:EDGE:SLOPe {POSitive|NEGative}
```

The :TRIGger:EDGE:SLOPe command sets the slope of the trigger source previously selected by the :TRIGger:EDGE:SOURce command. The LINE source has no slope.

**Query**

```
:TRIGger:EDGE:SLOPe?
```

The query returns the currently selected slope for the specified edge trigger source.

**Returned Format**

```
[:TRIGger:EDGE:SLOPe] {POS|NEG}<NL>
```

# EDGE:SOURce

Command

```
:TRIGger:EDGE:SOURce {CHANnel<N>|AUX|LINE|EXTernal}
```

The :TRIGger:EDGE:SOURce command selects the source for edge mode triggering. This is the source that will be used for subsequent :TRIGger:EDGE:SLOPe commands or queries.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**EXTernal and AUXiliary are Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

AUXiliary is only available in 4-channel Infiniium Oscilloscope models (including the 54815A, 54825A, 4835A, and 54845A).

Query

```
:TRIGger:EDGE:SOURce?
```

The query returns the currently selected edge mode trigger source.

Returned Format

```
[:TRIGger:EDGE:SOURce] {CHANnel<N>|AUX|LINE|EXTernal}<NL>
```

Use the Glitch Trigger Mode to find pulses in a waveform that are narrower than the rest of the pulses in the waveform.

To look for pulses that are wider than the other pulses in your waveform, you should use the pulse width trigger. Pulse width trigger is in the Advanced trigger menu under Violation trigger.

The oscilloscope identifies a glitch trigger by looking for a pulse that is narrower than other pulses in your waveform. You specify the width that the pulse must be narrower than, and the pulse polarity (positive or negative) that the oscilloscope should consider to be a glitch. For a positive glitch, the oscilloscope triggers when the falling edge of a pulse crosses the trigger level. For a negative glitch, the oscilloscope triggers when the rising edge of the pulse crosses the trigger level.

`Source` Use this control to select the oscilloscope channel used to trigger the oscilloscope.

`Level` Use the Level control to set the trigger level through which the glitch must pass before the oscilloscope will trigger.

When setting the trigger level for your waveform, it is usually best to choose a voltage value that is equal to the voltage value at the mid point of your waveform. For example, if you have a waveform with a minimum value of 0 (zero) volts and a maximum value of 5 volts, then 2.5 volts is the best place to set your trigger level. The reason this is the best choice is that there may be some ringing or noise at both the 0-volt and 5-volt levels that can cause false triggers.

When you adjust the trigger level control, a horizontal dashed line with a T on the right-hand side appears, showing you where the trigger level is with respect to your waveform. After a period of time the dashed line will disappear. To redisplay the line, adjust the trigger level control again, or activate the Trigger dialog. A permanent icon with arrow (either T, $T_L$, or $T_H$) is also displayed on the right side of the waveform area, showing the trigger level.

`Polarity` Use the Positive control to look for positive glitches. Use the Negative control to look for negative glitches.

`Width` Use the Width control to define the maximum pulse width that is considered a glitch. The glitch width range is from 1.5 ns to 160 ms.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

**Set the Mode Before Executing Commands**

Before you can execute the :TRIGger:GLITch commands, set the mode by entering:

```
:TRIGger:MODE GLITch
```

This command sets the conditions for the glitch polarity, source, and width trigger commands.

To query the oscilloscope for the trigger mode, enter:

```
:TRIGger:MODE?
```

You set up the :TRIGger:GLITch commands with the following commands and queries:

- POLarity
- SOURce
- WIDTh

## GLITch:POLarity

Command

```
:TRIGger:GLITch:POLarity {POSitive|NEGative}
```

This command defines the polarity of the glitch as positive or negative. The trigger source must be set using the :TRIGger:GLITch:SOURce command.

Query

```
:TRIGger:GLITch:POLarity?
```

The query returns the currently selected glitch polarity.

Returned Format

```
[:TRIGger:GLITch:POLarity] {POS|NEG}<NL>
```

## GLITch:SOURce

Command

```
:TRIGger:GLITch:SOURce {CHANnel<N>|EXTernal}
```

This command sets the source for the glitch trigger mode.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

Query

```
:TRIGger:GLITch:SOURce?
```

The query returns the currently selected source for the glitch trigger mode.

Returned Format

```
[:TRIGger:GLITch:SOURce] {CHANnel<N>|EXTernal}<NL>
```

# GLITch:WIDTh

**Command**

```
:TRIGger:GLITch:WIDTh <width>
```

This command sets the glitch width. The oscilloscope will trigger on a pulse that has a width less than the specified width.

`<width>` A real number for the glitch width, ranging from 1.5 ns to 160 ms.

**Query**

```
:TRIGger:GLITch:WIDTh?
```

The query returns the currently specified glitch width.

**Returned Format**

```
[:TRIGger:GLITch:WIDTh] <width><NL>
```

Use the COMM Trigger Mode to find a serial pattern of bits in a waveform. The COMM Trigger Mode is primarily used to find an isolated logically one bit in a waveform for mask testing applications. The pattern is defined by the standards used by the telecommunication and data communication industries. Mask testing is used to verify a waveform meets industrial standards which guarantees that equipment made by different manufacturers will work together.

**Set the Mode Before Executing Commands**

Before you can execute the :TRIGger:ADVanced:COMMunications commands, mask testing must be enabled at least one time. The :MTESt:ENABle command enables or disables mask testing. Then you can set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE COMM
```

To query the oscilloscope for the advanced trigger mode, enter:

```
:TRIGger:ADVanced:MODE?
```

The :TRIGger:ADVanced:COMM commands define the

# COMM:BWIDth

Command

```
:TRIGger:ADVanced:COMM:BWIDth <bwidth_value>
```

The :TRIGger:ADVanced:COMM:BWIDth command is used to set the width of a bit for your waveform. The bit width is usually defined in the mask standard for your waveform.

`<bwidth_value>` A real number that represents the width of a bit.

Query

```
:TRIGger:ADVanced:COMM:BWIDth?
```

The query returns the current bit width.

Returned Format

```
[:TRIGger:ADVanced:COMM:BWIDth] <bwidth_value><NL>
```

## COMM:ENCode

**Command**

```
:TRIGger:ADVanced:COMM:ENCode {RZ | NRZ}
```

This :TRIGger:ADVanced:COMM:ENCode command sets the type of waveform encoding for your waveform. You should use NRZ for CMI type waveforms and RZ for all other type of waveforms.

**Query**

```
:TRIGger:ADVanced:COMM:ENCode?
```

The :TRIGger:ADVanced:COMM:ENCode? query returns the current value of encoding

**Returned Format**

```
[:TRIGger:ADVanced:COMM:ENCode] {RZ | NRZ}<NL>
```

## COMM:LEVel

Command

```
:TRIGger:ADVanced:COMM:LEVel CHANnel<N>,<level>
```

The `:TRIGger:ADVanced:COMM:LEVel` command sets the voltage level used to determine a logic 1 from a logic 0 for the communication pattern.

`<N>` An integer, 1-2, for 54810/54820 Infiniium Oscilloscopes
An integer, 1-4, for all other Infiniium Oscilloscope models.

`<level>` A real number which is the logic level voltage.

Query

```
:TRIGger:ADVanced:COMM:LEVel?
```

The `:TRIGger:ADVanced:COMM:LEVel?` query returns the current level for the communication pattern.

Returned Format

```
[:TRIGger:ADVanced:COMM:LEVel] <level><NL>
```

# COMM:PATTern

Command

```
:TRIGger:ADVanced:COMM:PATTern
<bit>[,<bit[,<bit[,<bit[,<bit[,<bit]]]]]
```

The :TRIGger:ADVanced:COMM:PATTern command sets the pattern used for triggering the oscilloscope when in communication trigger mode. The pattern can be up to 6 bits long. For NRZ type waveforms with positive polarity, there must be at least one logic 0 to logic 1 transition in the pattern. For NRZ waveforms with negative polarity there must be at least one logic 1 to logic 0 transition in the pattern. For RZ type waveforms the pattern must have at least one logic 1 bit for positive polarity. For RZ type waveforms the pattern must have at least one logic -1 bit for negative polarity.

`<bit>` A 1, -1, or 0.

Query

```
:TRIGger:ADVanced:COMM:PATTern?
```

The `:TRIGger:ADVanced:COMM:PATTern?` query returns the current communication trigger pattern.

Returned Format

```
[:TRIGger:ADVanced:COMM:PATTern] <pattern><NL>
```

`<pattern>` A string of up to 6 characters.

# COMM:POLarity

Command

```
:TRIGger:ADVanced:COMM:POLarity {POSitive | NEGative}
```

The `:TRIGger:ADVanced:COMM:POLarity` command directly controls the trigger slope used for communication trigger. When set to a positive value, the rising edge of a pulse or waveform is used to trigger the oscilloscope. When set to a negative value, the falling edge of a pulse or waveform is used.

The polarity setting is also used to check for valid patterns. If you are trying to trigger on an isolated 1 pattern, you should set the polarity to positive. If you are trying to trigger on an isolated -1 pattern, you should set the polarity to negative.

Query

```
:TRIGger:ADVanced:COMM:POLarity?
```

The `:TRIGger:ADVanced:COMM:POLarity?` query returns the current setting for polarity.

Returned Format

```
[:TRIGger:ADVanced:COMM:POLarity} {1|0}<NL>
```

## COMM:SOURce

Command

```
:TRIGger:ADVanced:COMM:SOURce CHANnel<N>
```

The :TRIGger:ADVanced:COMM:SOURce command selects the channel used for the communication trigger.

<N> An integer, 1-2, for 54810/54820 Infiniium Oscilloscopes
An integer, 1-4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:COMM:SOURce?
```

The `:TRIGger:ADVanced:COMM:SOURce?` query returns the currently selected communication trigger source.

Returned Format

```
[:TRIGger:ADVanced:COMM:SOURce] CHANnel<N><NL>
```

Logic triggering is similar to the way that a logic analyzer captures data. This mode is useful when you are looking for a particular set of ones and zeros on a computer bus or control lines. You determine which channels the oscilloscope uses to form the trigger pattern. Because you can set the voltage level that determines a logic 1 or a logic 0, any logic family that you are probing can be captured.

There are two types of logic triggering: Pattern and State. The difference between pattern and state triggering modes is that state triggering uses one of the oscilloscope channels as a clock.

Use pattern triggering to trigger the oscilloscope using more than one channel as the trigger source. You can also use pattern triggering to trigger on a pulse of a given width.

The Pattern Trigger Mode identifies a trigger condition by looking for a specified pattern. A pattern is a logical combination of the channels. Each channel can have a value of High (H), Low (L) or Don't Care (X). A value is considered a High when your waveform's voltage level is greater than its trigger level, and a Low when the voltage level is less than its trigger level. If a channel is set to Don't Care, it is not used as part of the pattern criteria.

One additional qualifying condition determines when the oscilloscope triggers once the pattern is found. The :PATTern:CONDition command has five possible ways to qualify the trigger:

`Entered` The oscilloscope will trigger on the edge of the source that makes the pattern true.

`Exited` The oscilloscope will trigger on the edge of the source that makes the pattern false.

`Present >` The oscilloscope will trigger when the pattern is present for greater than the time that you specify. An additional parameter allows the oscilloscope to trigger when the pattern goes away or when the time expires.

`Present <` The oscilloscope will trigger when the pattern is present for less than the time that you specify.

`Range` The oscilloscope will trigger on the edge of the waveform that makes the pattern invalid as long as the pattern is present within the range of times that you specify.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

**Set the Mode Before Executing Commands**

Before you can execute the :TRIGger:ADVanced:PATTern commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE PATTern
```

To query the oscilloscope for the advanced trigger mode, enter:

```
:TRIGger:ADVanced:MODE?
```

The :TRIGger:ADVanced:PATTern commands define the conditions for the Pattern Trigger Mode. As described in the following commands, you set up the :TRIGger:ADVanced:PATTern commands with the following commands and queries:

- CONDition
- LOGic

## PATTern:CONDition

Command

```
:TRIGger:ADVanced:PATTern:CONDition {ENTered|EXITed
| {GT,<time>[,PEXits|TIMeout]} |
{LT,<time>} | {RANGe,<gt_time>,<lt_time>}}
```

This command describes the condition applied to the trigger pattern to actually generate a trigger.

`<gt_time>` The minimum time (greater than time) for the trigger pattern, from 20 ns to 160 ms.

`<lt_time>` The maximum time (less than time) for the trigger pattern, from 30 ns to 160 ms.

`<time>` The time condition, in seconds, for the pattern trigger, from 1.5 ns to 160 ms.

With the greater than parameter, Pattern Exits (PEXits) or TIMeout controls when the trigger is generated.

Query

```
:TRIGger:ADVanced:PATTern:CONDition?
```

The query returns the currently defined trigger condition.

Returned Format

```
[:TRIGger:ADVanced:PATTern:CONDition] {ENTered|EXITed |
{GT,<time>[,PEXits|TIMeout]} | {LT,<time>} |
{RANGe,<gt_time>, <lt_time>}}<NL>
```

# PATTern:LOGic

Command

```
:TRIGger:ADVanced:PATTern:LOGic
{{CHANnel<N>|EXTernal},{HIGH|LOW|DONTcare}}
```

This command defines the logic criteria for a selected channel.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:PATTern:LOGic?
{CHANnel<N>|EXTernal}
```

The query returns the current logic criteria for a selected channel.

Returned Format

```
[:TRIGger:ADVanced:PATTern:LOGic {CHANnel<N>|EXTernal}]
{HIGH|LOW|DONTcare}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

Logic triggering is similar to the way that a logic analyzer captures data. This mode is useful when you are looking for a particular set of ones and zeros on a computer bus or control lines. You determine which channels the oscilloscope uses to form the trigger pattern. Because you can set the voltage level that determines a logic 1 or a logic 0, any logic family that you are probing can be captured.

There are two types of logic triggering: Pattern and State. The difference between pattern and state triggering modes is that state triggering uses one of the oscilloscope channels as a clock.

Use state triggering when you want the oscilloscope to use several channels as the trigger source, with one of the channels being used as a clock waveform.

The State trigger identifies a trigger condition by looking for a clock edge on one channel and a pattern on the remaining channels. A pattern is a logical combination of the remaining channels. Each channel can have a value of High (H), Low (L) or Don't Care (X). A value is considered a High when your waveform's voltage level is greater than the trigger level and a Low when the voltage level is less than the trigger level. If a channel is set to Don't Care, it is not used as part of the pattern criteria. You can select the clock edge as either rising or falling.

The logic type control determines whether or not the oscilloscope will trigger when the specified pattern is found on a clock edge. When AND is selected, the oscilloscope will trigger on a clock edge when input waveforms match the specified pattern. When NAND is selected, the oscilloscope will trigger when the input waveforms are different from the specified pattern and a clock edge occurs.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

**Set the Mode Before Executing Commands**

Before you can execute the :TRIGger:ADVanced:STATe commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE STATe
```

To query the oscilloscope for the advanced trigger mode, enter:

```
:TRIGger:ADVanced:MODE?
```

The :TRIGger:ADVanced:STATe commands define the conditions for the State Trigger Mode. As described in the following commands, you set up the :TRIGger:ADVanced:STATe commands with the following commands and queries:

- CLOCk
- CONDition
- LOGic
- LTYPe
- SLOPe

## STATe:CLOCk

Command

```
:TRIGger:ADVanced:STATe:CLOCk {CHANnel<N>|EXTernal}
```

This command selects the source for the clock waveform in the State Trigger Mode.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:STATe:CLOCk?
```

The query returns the currently selected clock source.

Returned Format

```
[:TRIGger:ADVanced:STATe:CLOCk] {CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

## STATe:CONDition

**Command**

```
:TRIGger:ADVanced:STATe:CONDition {TRUE|FALSe}
```

This command determines if a trigger is generated when the pattern specified by the LOGic command is TRUE (present for AND pattern) or FALSe (not present—NAND). This command is the same as :STATe:LTYPe.

**Query**

```
:TRIGger:ADVanced:STATe:CONDition?
```

The query returns the currently specified condition.

**Returned Format**

```
[:TRIGger:ADVanced:STATe:CONDition] {TRUE|FALSe}<NL>
```

## STATe:LOGic

Command

```
:TRIGger:ADVanced:STATe:LOGic
{{CHANnel<N>|EXTernal},{LOW|HIGH|DONTcare}}
```

This command defines the logic state of the specified channel for the State Trigger Mode.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:STATe:LOGic?
{CHANnel<N>|EXTernal}
```

The query returns the logic state definition for the specified channel input.

<N> N is the channel number, an integer in the range of 1 - 4.

Returned Format

```
[:TRIGger:ADVanced:STATe:LOGic {CHANnel<N>|EXTernal}]
{LOW|HIGH|DONTcare}<NL>
```

DONTcare is returned on the channel currently selected as clock.

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

## STATe:LTYPe

**Command**

```
:TRIGger:ADVanced:STATe:LTYPe {AND|NAND}
```

This command defines the state trigger logic type. This command is the same as :STATe:CONDition.

**Query**

```
:TRIGger:ADVanced:STATe:LTYPe?
```

The query returns the currently specified state trigger logic type.

**Returned Format**

```
[:TRIGger:ADVanced:STATe:LTYPe] {AND|NAND}<NL>
```

## STATe:SLOPe

Command

```
:TRIGger:ADVanced:STATe:SLOPe {POSitive|NEGative}
```

This command specifies the slope of the input previously selected by the :TRIGger:ADVanced:STATe:CLOCk command.

Query

```
:TRIGger:ADVanced:STATe:SLOPe?
```

The query returns the currently defined slope for the clock in State Trigger Mode.

Returned Format

```
[:TRIGger:ADVanced:STATe:SLOPe] {POSitive|NEGative}<NL>
```

You can set the delay mode to delay by events or time. Use Delay By Event mode to view pulses in your waveform that occur a number of events after a specified waveform edge. Infiniium Oscilloscopes identify a trigger by arming on the edge you specify, counting a number of events, then triggering on the specified edge.

`Arm On` Use Arm On to set the source, level, and slope for arming the trigger circuitry. When setting the arm level for your waveform, it is usually best to choose a voltage value that is equal to the voltage value at the mid point of your waveform. For example, if you have a waveform with a minimum value of 0 (zero) volts and a maximum value of 5 volts, then 2.5 volts is the best place to set your arm level. The reason this is the best choice is that there may be some ringing or noise at both the 0-volt and 5-volt levels that can cause false triggers.

When you adjust the arm level control, a horizontal dashed line with a T on the right-hand side appears showing you where the arm level is with respect to your waveform. After a period of time the dashed line will disappear. To redisplay the line, adjust the arm level control again, or activate the Trigger dialog.

`Delay By Event` Use Delay By Event to set the source, level, and edge to define an event. When setting the event level for your waveform, it is usually best to choose a voltage value that is equal to the voltage value at the mid point of your waveform. For example, if you have a waveform with a minimum value of 0 (zero) volts and a maximum value of 5 volts, then 2.5 volts is the best place to set your event level. The reason this is the best choice is that there may be some ringing or noise at both the 0-volt and 5-volt levels that can cause false triggers.

`Event` Use Event to set the number of events (edges) that must occur after the oscilloscope is armed until it starts to look for the trigger edge.

`Trigger On` Use Trigger On to set the trigger source and trigger slope required to trigger the oscilloscope. Each source can have only one level, so if you are arming and triggering on the same source, only one level is used.

**Set the Mode Before Executing Commands**

Before you can execute the :TRIGger:ADVanced:DELay commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE DELay
```

The ADVanced DELay commands define the conditions for the Delay Trigger Mode. The Delay By Events Mode lets you view pulses in your waveform that occur a number of events after a specified waveform edge. After entering the commands above, to select Delay By Events Mode, enter:

```
:TRIGger:ADVanced:DELay:MODE EDLY
```

Then you can use the Event Delay (EDLY) commands and queries for ARM, EVENt, and TRIGger on the following pages.

To query the oscilloscope for the advanced trigger mode or the advanced trigger delay mode, enter:

```
:TRIGger:ADVanced:MODE?  or
:TRIGger:ADVanced:DELay:MODE?
```

# EDLY:ARM:SOURce

Command

```
:TRIGger:ADVanced:DELay:EDLY:ARM:SOURce
{CHANnel<N>|EXTernal}
```

This command sets the Arm On source for arming the trigger circuitry when the oscilloscope is in the Delay By Event trigger mode.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:DELay:EDLY:ARM:SOURce?
```

The query returns the currently defined Arm On source for the Delay By Event trigger mode.

Returned Format

```
[:TRIGger:ADVanced:DELay:EDLY:ARM:SOURce]
{CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# EDLY:ARM:SLOPe

Command

```
:TRIGger:ADVanced:DELay:EDLY:ARM:SLOPe
{NEGative|POSitive}
```

This command sets a positive or negative slope for arming the trigger circuitry when the oscilloscope is in the Delay By Event trigger mode.

Query

```
:TRIGger:ADVanced:DELay:EDLY:ARM:SLOPe?
```

The query returns the currently defined slope for the Delay By Event trigger mode.

Returned Format

```
[:TRIGger:ADVanced:DELay:EDLY:ARM:SLOPe]
{NEGative|POSitive}<NL>
```

## EDLY:EVENt:DELay

**Command**

```
:TRIGger:ADVanced:DELay:EDLY:EVENt:DELay
<edge_number>
```

This command sets the event count for a Delay By Event trigger event.

`<edge_num>` An integer from 0 to 16,000,000 specifying the number of edges to delay.

**Query**

```
:TRIGger:ADVanced:DELay:EDLY:EVENt:DELay?
```

The query returns the currently defined number of events to delay before triggering on the next Trigger On condition in the Delay By Event trigger mode.

**Returned Format**

```
[:TRIGger:ADVanced:DELay:EDLY:EVENt:DELay]
<edge_number><NL>
```

# EDLY:EVENt:SOURce

Command

```
:TRIGger:ADVanced:DELay:EDLY:EVENt:SOURce
{CHANnel<N>|EXTernal}
```

This command sets the Event source for a Delay By Event trigger event.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:DELay:EDLY:EVENt:SOURce?
```

The query returns the currently defined Event source in the Delay By Event trigger mode.

Returned Format

```
[:TRIGger:ADVanced:DELay:EDLY:EVENt:SOURce]
{CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

## EDLY:EVENt:SLOPe

**Command**

```
:TRIGger:ADVanced:DELay:EDLY:EVENt:SLOPe
{NEGative|POSitive}
```

This command sets the trigger slope for the Delay By Event trigger event.

**Query**

```
:TRIGger:ADVanced:DELay:EDLY:EVENt:SLOPe?
```

The query returns the currently defined slope for an event in the Delay By Event trigger mode.

**Returned Format**

```
[:TRIGger:ADVanced:EDLY:EVENt:SLOPe]
{NEGative|POSitive}<NL>
```

# EDLY:TRIGger:SOURce

Command

```
:TRIGger:ADVanced:DELay:EDLY:TRIGger:SOURce
{CHANnel<N>|EXTernal}
```

This command sets the Trigger On source for a Delay By Event trigger event.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

Query

```
:TRIGger:ADVanced:DELay:EDLY:TRIGger:SOURce?
```

The query returns the currently defined Trigger On source for the event in the Delay By Event trigger mode.

Returned Format

```
[:TRIGger:ADVanced:DELay:EDLY:TRIGger:SOURce]
{CHANnel<N>|EXTernal}<NL>
```

## EDLY:TRIGger:SLOPe

**Command**

```
:TRIGger:ADVanced:DELay:EDLY:TRIGger:SLOPe
{NEGative|POSitive}
```

This command sets the trigger slope for the Delay By Event trigger event.

**Query**

```
:TRIGger:ADVanced:DELay:EDLY:TRIGger:SLOPe?
```

The query returns the currently defined slope for an event in the Delay By Event trigger mode.

**Returned Format**

```
[:TRIGger:ADVanced:DELay:EDLY:TRIGger:SLOPe]
{NEGative|POSitive}<NL>
```

You can set the delay mode to delay by events or time. Use Delay By Time mode to view pulses in your waveform that occur a long time after a specified waveform edge. The Delay by Time identifies a trigger condition by arming on the edge you specify, waiting a specified amount of time, then triggering on a specified edge. This can be thought of as two-edge triggering, where the two edges are separated by a selectable amount of time.

It is also possible to use the Horizontal Position control to view a pulse some period of time after the trigger has occurred. The problem with this method is that the further the pulse is from the trigger, the greater the possibility that jitter will make it difficult to view. Delay by Time eliminates this problem by triggering on the edge of interest.

`Arm On` Use Arm On to set the source, level, and slope for the arming condition. When setting the arm level for your waveform, it is usually best to choose a voltage value that is equal to the voltage value at the mid point of your waveform. For example, if you have a waveform with a minimum value of 0 (zero) volts and a maximum value of 5 volts, then 2.5 volts is the best place to set your arm level. The reason this is the best choice is that there may be some ringing or noise at both the 0-volt and 5-volt levels that can cause false triggers.

When you adjust the arm level control, a horizontal dashed line with a T on the right-hand side appears showing you where the arm level is with respect to your waveform. After a period of time the dashed line will disappear. To redisplay the line, adjust the arm level control again, or activate the Trigger dialog.

`Delay By Time` Use Delay By Time to set the amount of delay time from when the oscilloscope is armed until it starts to look for the trigger edge. The range is from 30 ns to 160 ms.

`Trigger On` Use Trigger On to set the source and slope required to trigger the oscilloscope. Trigger On Level is slaved to Arm On Level.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

**Set the Mode Before Executing Commands**

Before you can execute the :TRIGger:ADVanced:DELay commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE DELay
```

The ADVanced DELay commands define the conditions for the Delay Trigger Mode. The Delay By Time Mode lets you view pulses in your waveform that occur a specified time after a specified waveform edge. After entering the commands above, to select Delay By Time Mode, enter:

```
:TRIGger:ADVanced:DELay:MODE TDLY
```

Then you can use the Time Delay (TDLY) commands and queries for ARM, DELay, and TRIGger on the following pages.

To query the oscilloscope for the advanced trigger mode or the advanced trigger delay mode, enter:

```
:TRIGger:ADVanced:MODE?  or
:TRIGger:ADVanced:DELay:MODE?
```

# TDLY:ARM:SOURce

Command

```
:TRIGger:ADVanced:DELay:TDLY:ARM:SOURce
{CHANnel<N>|EXTernal}
```

This command sets the Arm On source for arming the trigger circuitry when the oscilloscope is in the Delay By Time trigger mode.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:DELay:TDLY:ARM:SOURce?
```

The query returns the currently defined channel source for the Delay By Time trigger mode.

Returned Format

```
[:TRIGger:ADVanced:DELay:TDLY:ARM:SOURce]
{CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# TDLY:ARM:SLOPe

**Command**

```
:TRIGger:ADVanced:DELay:TDLY:ARM:SLOPe
{NEGative|POSitive}
```

This command sets a positive or negative slope for arming the trigger circuitry when the oscilloscope is in the Delay By Time trigger mode.

**Query**

```
:TRIGger:ADVanced:DELay:TDLY:ARM:SLOPe?
```

The query returns the currently defined slope for the Delay By Time trigger mode.

**Returned Format**

```
[:TRIGger:ADVanced:DELay:TDLY:ARM:SLOPe]
{NEGative|POSitive}<NL>
```

# TDLY:DELay

**Command**

```
:TRIGger:ADVanced:DELay:TDLY:DELay <delay>
```

This command sets the delay for a Delay By Time trigger event.

`<delay>` Time, in seconds, set for the delay trigger, from 30 ns to 160 ms.

**Query**

```
:TRIGger:ADVanced:DELay:TDLY:DELay?
```

The query returns the currently defined time delay before triggering on the next Trigger On condition in the Delay By Time trigger mode.

**Returned Format**

```
[:TRIGger:ADVanced:DELay:TDLY:DELay] <delay><NL>
```

## TDLY:TRIGger:SOURce

Command

```
:TRIGger:ADVanced:DELay:TDLY:TRIGger:SOURce
{CHANnel<N>|EXTernal}
```

This command sets the Trigger On source for a Delay By Time trigger event.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:DELay:TDLY:TRIGger:SOURce?
```

The query returns the currently defined Trigger On source in the Delay By Time trigger mode.

Returned Format

```
[:TRIGger:ADVanced:DELay:TDLY:TRIGger:SOURce]
{CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# TDLY:TRIGger:SLOPe

Command

```
:TRIGger:ADVanced:DELay:TDLY:TRIGger:SLOPe
{NEGative|POSitive}
```

This command sets the trigger slope for the Delay By Time trigger event.

Query

```
:TRIGger:ADVanced:DELay:TDLY:TRIGger:SLOPe?
```

The query returns the currently defined slope for an event in the Delay By Time trigger mode.

Returned Format

```
[:TRIGger:ADVanced:DELay:TDLY:TRIGger:SLOPe]
{NEGative|POSitive}<NL>
```

Use TV trigger mode to trigger on one of the standard television waveforms. Also, use this mode to trigger on a custom television waveform that you define, as described in the next section.

There are four types of television (TV) trigger modes: 525 (NTSC or PAL-M), 625 (PAL or SECAM), 875 (High Definition Zenith Standard), and User Defined. The 525, 625, and 875 are predefined video standards used throughout the world. The User Defined TV trigger, described in the next section, lets you trigger on nonstandard TV waveforms.

**525, 625, and 875 TV Trigger Modes**

`Source` Use the Source control to select one of the oscilloscope channels as the trigger source.

`Level` Use to set the trigger voltage level. When setting the trigger level for your waveform, it is usually best to choose a voltage value that is just below the bottom of burst.

When you adjust the trigger level control, a horizontal dashed line with a T on the right-hand side appears showing you where the trigger level is with respect to your waveform. After a period of time the dashed line will disappear. To redisplay the line, adjust the trigger level control again, or activate the Trigger dialog.

`Positive or Negative Sync` Use the Positive and Negative Sync controls to select either a positive sync pulse or a negative sync pulse as the trigger.

`Field` Use the Field control to select video field 1 or video field 2 as the trigger.

`Line` Use the Line control to select the horizontal line you want to view within the chosen video field.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

**STV Commands**

These commands set the conditions for the TV trigger mode using standard, predefined parameters (in STV mode), or user-defined parameters (in UDTV mode). The STV commands are used for triggering on television waveforms, and let you select one of the TV waveform frames and one of the lines within that frame.

**Set the Mode Before Executing Commands**

Before executing the :TRIGger:ADVanced:STV commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE TV  and

:TRIGger:ADVanced:TV:MODE L525  or
:TRIGger:ADVanced:TV:MODE L625  or
:TRIGger:ADVanced:TV:MODE L875
```

To query the oscilloscope for the advanced trigger mode or the advanced trigger TV mode, enter:

```
:TRIGger:ADVanced:MODE?  or
:TRIGger:ADVanced:TV:MODE?
```

You set up the :TRIGger:ADVanced:TV:STV commands with the following commands and queries:

- FIELd
- LINE
- SOURce
- SPOLarity

## STV:FIELd

**Command**

```
:TRIGger:ADVanced:TV:STV:FIELd {1|2}
```

This command is available in standard TV trigger modes L525, L625, and L875.

The :TRIGger:ADVanced:TV:STV:FIELd command selects which TV waveform field is used during standard TV trigger mode. The line within the selected field is specified using the :TRIGger:ADVanced:TV:STV:LINE <line_number> command.

**Query**

```
:TRIGger:ADVanced:TV:STV:FIELd?
```

The query returns the current television waveform field.

**Returned Format**

```
[:TRIGger:ADVanced:TV:STV:FIELd] {1|2}<NL>
```

# STV:LINE

Command

```
:TRIGger:ADVanced:TV:STV:LINE <line_number>
```

This command is available in standard TV trigger modes L525, L625, and L875.

The :TRIGger:ADVanced:TV:STV:LINE command selects the horizontal line that the instrument will trigger on. Allowable line_number entry depends on the :TRIGger:ADVanced:TV:STV:FIELd selected. Once the vertical sync pulse of the selected field is received, the trigger is delayed by the number of lines specified.

`<line_number>` Horizontal line number. Allowable values range from 1 to 875, depending on :TRIGger:ADVanced:TV:STV:FIELd settings as shown below.

| | **STV Modes** | | |
|---|---|---|---|
| | **525** | **625** | **875** |
| **Field 1** | 1 to 263 | 1 to 313 | 1 to 438 |
| **Field 2** | 1 to 262 | 314 to 625 | 1 to 437 |

Query

```
:TRIGger:ADVanced:TV:STV:LINE?
```

The query returns the current line number.

Returned Format

```
[:TRIGger:ADVanced:TV:STV:LINE] <line_number><NL>
```

# STV:SOURce

Command

```
:TRIGger:ADVanced:TV:STV:SOURce
{CHANnel<N>|EXTernal}
```

This command is available in standard TV trigger modes L525, L625, and L875.

The :TRIGger:ADVanced:TV:STV:SOURce command selects the source for standard TV mode triggering. This is the source that will be used for subsequent :TRIGger:ADVanced:TV:STV commands and queries.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:TV:STV:SOURce?
```

The query returns the currently selected standard TV trigger mode source.

Returned Format

```
[:TRIGger:ADVanced:TV:STV:SOURce] {CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

## STV:SPOLarity

Command

```
:TRIGger:ADVanced:TV:STV:SPOLarity
{NEGative|POSitive}
```

This command is available in standard TV trigger modes L525, L625, and L875. The :TRIGger:ADVanced:TV:STV:SPOLarity (Sync POLarity) command specifies the vertical sync pulse polarity for the selected field used during standard TV mode triggering.

Query

```
:TRIGger:ADVanced:TV:STV:SPOLarity?
```

The query returns the currently selected sync pulse polarity.

Returned Format

```
[:TRIGger:ADVanced:TV:STV:SPOLarity] {NEGative|POSitive}<NL>
```

Use TV trigger mode to trigger on one of the standard television waveforms, as described in the previous section, and to trigger on a custom television waveform that you define. The User Defined TV trigger lets you trigger on nonstandard TV waveforms.

**User Defined TV Trigger**

`Source` Use the Source control to select one of the oscilloscope channels as the trigger source.

`Level` Use the Level control to set the trigger voltage level.

When setting the trigger level for your waveform, it is usually best to choose a voltage value that is just below the bottom of burst.

When you adjust the trigger level control, a horizontal dashed line with a T on the right-hand side appears showing you where the trigger level is with respect to your waveform. After a period of time the dashed line will disappear. To redisplay the line, adjust the trigger level control again, or activate the Trigger dialog. A permanent icon with arrow (either T, $T_L$, or $T_H$) is also displayed on the right side of the waveform area, showing the trigger level.

`Pos or Neg` Use the Pos and Neg controls to select either a positive pulse or a negative pulse to arm the trigger circuitry.

`Time >` Use the Time > control to set the minimum time that the pulse must be present to be considered a valid sync pulse.

`Time <` Use the Time < control to set the maximum time that the pulse must be present to be considered a valid sync pulse.

`Trigger On` Use the Trigger On control to select either a Rising or Falling edge as the trigger condition.

`Edge Number` Use the Edge Number control to select the number of edges you want the oscilloscope to count before triggering.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

**UDTV Commands**

These commands set the conditions for the TV trigger mode using user-defined parameters. They are used for triggering on non-standard television waveforms, and let you define the conditions that must be met before a trigger occurs.

**Set the Mode Before Executing Commands**

Before executing the :TRIGger:ADVanced:TV:UDTV commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE TV  and
:TRIGger:ADVanced:TV:MODE UDTV
```

To query the oscilloscope for the advanced trigger mode or the advanced trigger TV mode, enter:

```
:TRIGger:ADVanced:MODE?  or
:TRIGger:ADVanced:TV:MODE?
```

You set up the :TRIGger:ADVanced:TV:UDTV commands with the following commands and queries:

- EDGE
- ENUMber
- PGTHan
- PLTHan
- POLarity
- SOURce

When triggering for User Defined TV mode:

- Set the channel or trigger source for the trigger using:

  ```
  :TRIGger:ADVanced:TV:UDTV:SOURce
  ```

- Set the conditions for arming the trigger using:

  ```
  :TRIGger:ADVanced:TV:UDTV:PGTHan,
  :TRIGger:ADVanced:TV:UDTV:PLTHan, and
  :TRIGger:ADVanced:TV:UDTV:POLarity.
  ```

- Set the number of events to delay after the trigger is armed using:

  ```
  :TRIGger:ADVanced:TV:UDTV:ENUMber
  ```

- Set the waveform edge that causes the trigger to occur after arming and delay using:

  ```
  :TRIGger:ADVanced:TV:UDTV:EDGE
  ```

# UDTV:EDGE

Command

```
:TRIGger:ADVanced:TV:UDTV:EDGE {RISing|FALLing}
```

The :TRIGger:ADVanced:TV:UDTV:EDGE command specifies the waveform edge that causes a trigger to occur after arming and delay conditions have been satisfied.

Query

```
:TRIGger:ADVanced:TV:UDTV:EDGE?
```

The query returns the currently selected Trigger On edge selection.

Returned Format

```
[:TRIGger:ADVanced:TV:UDTV:EDGE] {RISing|FALLing}<NL>
```

# UDTV:ENUMber

**Command**

```
:TRIGger:ADVanced:TV:UDTV:ENUMber <count>
```

The :TRIGger:ADVanced:TV:UDTV:ENUMber command specifies the number of events (horizontal sync pulses) to delay after arming the trigger before looking for the trigger event. Specify conditions for arming the trigger using:

TRIGger:ADVanced:TV:UDTV:PGTHan,

TRIGger:ADVanced:TV:UDTV:PLTHan, and

TRIGger:ADVanced:TV:UDTV:POLarity.

`<count>` An integer for the number of events to delay. Allowable values range from 1 to 16,000,000.

**Query**

```
:TRIGger:ADVanced:TV:UDTV:ENUMber?
```

The query returns the currently programmed count value.

**Returned Format**

```
[:TRIGger:ADVanced:TV:UDTV:ENUMber] <count><NL>
```

# UDTV:PGTHan

Command

```
:TRIGger:ADVanced:TV:UDTV:PGTHan <lower_limit>
```

The :TRIGger:ADVanced:TV:UDTV:PGTHan (Present Greater THan) command specifies the minimum pulse width of the waveform used to arm the trigger used during user-defined trigger mode.

`<lower_limit>` Minimum pulse width (time >), from 20 ns to 150 ms.

Query

```
:TRIGger:ADVanced:TV:UDTV:PGTHan?
```

The query returns the currently selected minimum pulse width.

Returned Format

```
[:TRIGger:ADVanced:TV:UDTV:PGTHan] <lower_limit><NL>
```

## UDTV:PLTHan

**Command** `:TRIGger:ADVanced:TV:UDTV:PLTHan <upper_limit>`

The :TRIGger:ADVanced:TV:UDTV:PLTHan (Present Less THan) command specifies the maximum pulse width of the waveform used to arm the trigger used during user-defined trigger mode.

`<upper_limit>` Maximum pulse width (time <), from 30 ns to 160 ms.

**Query** `:TRIGger:ADVanced:TV:UDTV:PLTHan?`

The query returns the currently selected maximum pulse width.

**Returned Format** `[:TRIGger:ADVanced:TV:UDTV:PLTHan] <upper_limit><NL>`

## UDTV:POLarity

Command

```
:TRIGger:ADVanced:TV:UDTV:POLarity
{NEGative|POSitive}
```

The :TRIGger:ADVanced:TV:UDTV:POLarity command specifies the polarity for the sync pulse used to arm the trigger in the user-defined trigger mode.

Query

```
:TRIGger:ADVanced:TV:UDTV:POLarity?
```

The query returns the currently selected UDTV sync pulse polarity.

Returned Format

```
[:TRIGger:ADVanced:TV:UDTV:POLarity]
{NEGative|POSitive}<NL>
```

# UDTV:SOURce

Command

```
:TRIGger:ADVanced:TV:UDTV:SOURce
{CHANnel<N>|EXTernal}
```

The :TRIGger:ADVanced:TV:UDTV:SOURce command selects the source for user-defined TV mode triggering. This is the source that will be used for subsequent :TRIGger:ADVanced:TV:UDTV commands and queries.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:TV:UDTV:SOURce?
```

The query returns the currently selected user-defined TV trigger mode source.

Returned Format

```
[:TRIGger:ADVanced:TV:UDTV:SOURce]
{CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

Violation triggering helps you find conditions within your circuit that violate the design rules. There are four types of violation triggering: Pulse Width, Setup and Hold Time, and Transition.

`PWIDth` This mode lets you find pulses that are wider than the rest of the pulses in your waveform. It also lets you find pulses that are narrower than the rest of the pulses in the waveform.

`SETup` This mode lets you find violations of setup and hold times in your circuit. Use this mode to select setup time triggering, hold time triggering, or both setup and hold time triggering.

`TRANsition` This mode lets you find any edge in your waveform that violates a rise time or fall time specification. The Infiniium oscilloscope can be set to trigger on rise times or fall times that are too slow or too fast.

# VIOLation:MODE

Command

```
:TRIGger:ADVanced:VIOLation:MODE {PWIDth | SETup |
TRANsition}
```

After you have selected the advanced trigger mode with the commands :TRIGger:MODE ADVanced and :TRIGger:ADVanced:MODE VIOLation, the :TRIGger:ADVanced:VIOLation:MODE <violation_mode> command specifies the mode for trigger violations. The <violation_mode> is either PWIDth, SETup, or TRANsition.

Query

```
:TRIGger:ADVanced:VIOLation:MODE?
```

The query returns the currently defined mode for trigger violations.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:MODE] {PWIDth | SETup |
TRANsition}<NL>
```

Use Pulse Width Violation Mode to find pulses that are wider than the rest of the pulses in your waveform. You can also use this mode to find pulses that are narrower than the rest of the pulses in the waveform.

The oscilloscope identifies a pulse width trigger by looking for a pulse that is either wider than or narrower than other pulses in your waveform. You specify the pulse width and pulse polarity (positive or negative) that the oscilloscope uses to determine a pulse width violation. For a positive polarity pulse, the oscilloscope triggers when the falling edge of a pulse crosses the trigger level. For a negative polarity pulse, the oscilloscope triggers when the rising edge of a pulse crosses the trigger level.

When looking for narrower pulses, pulse width less than (Width <) trigger is the same as glitch trigger.

`Source` Use Source to select the oscilloscope channel used to trigger the oscilloscope.

`Level` Use the Level control to set the voltage level through which the pulse must pass before the oscilloscope will trigger.

When setting the trigger level for your waveform, it is usually best to choose a voltage value that is equal to the voltage value at the mid point of your waveform. For example, if you have a waveform with a minimum value of 0 (zero) volts and a maximum value of 5 volts, then 2.5 volts is the best place to set your trigger level. The reason this is the best choice is that there may be some ringing or noise at both the 0-volt and 5-volt levels that can cause false triggers.

When you adjust the trigger level control, a horizontal dashed line with a T on the right-hand side appears showing you where the trigger level is with respect to your waveform. After a period of time the dashed line will disappear. To redisplay the line, adjust the trigger level control again, or activate the Trigger dialog. A permanent icon with arrow (either T, $T_L$, or $T_H$) is also displayed on the right side of the waveform area, showing the trigger level.

`Polarity` Use the Polarity control to specify positive or negative pulses.

`Direction` Use Direction to set whether a pulse must be wider (Width >) or narrower (Width <) than the width value to trigger the oscilloscope.

`Width` Use the Width control to define how wide of a pulse will trigger the oscilloscope. The glitch width range is from 1.5 ns to 160 ms.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

**Set the Mode Before Executing Commands**

Before executing the :TRIGger:ADVanced:VIOLation:PWIDth commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE VIOLation  and
:TRIGger:ADVanced:VIOLation:MODE PWIDth
```

To query the oscilloscope for the advanced trigger violation mode, enter:

```
:TRIGger:ADVanced:VIOLation:MODE?
```

# VIOLation:PWIDth:SOURce

Command

```
:TRIGger:ADVanced:VIOLation:PWIDth:SOURce
{CHANnel<N>|EXTernal}
```

This command specifies the channel source used to trigger the oscilloscope with a pulse width trigger. The level is the voltage through which the pulse must pass to trigger the oscilloscope.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the voltage through which the pulse must pass before the oscilloscope will trigger.

Query

```
:TRIGger:ADVanced:VIOLation:PWIDth:SOURce?
```

The query returns the currently defined channel source for the pulse width trigger.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:PWIDth:SOURce]
{CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

## VIOLation:PWIDth:POLarity

Command

```
:TRIGger:ADVanced:VIOLation:PWIDth:POLarity
{NEGative|POSitive}
```

This command specifies the pulse polarity that the oscilloscope uses to determine a pulse width violation. For a negative polarity pulse, the oscilloscope triggers when the rising edge of a pulse crosses the trigger level. For a positive polarity pulse, the oscilloscope triggers when the falling edge of a pulse crosses the trigger level.

Query

```
:TRIGger:ADVanced:VIOLation:PWIDth:POLarity?
```

The query returns the currently defined polarity for the pulse width trigger.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:PWIDth:POLarity]
{NEGative|POSitive}<NL>
```

## VIOLation:PWIDth:DIRection

**Command**

```
:TRIGger:ADVanced:VIOLation:PWIDth:DIRection
{GTHan|LTHan}
```

This command specifies whether a pulse must be wider or narrower than the width value to trigger the oscilloscope.

**Query**

```
:TRIGger:ADVanced:VIOLation:PWIDth:DIRection?
```

The query returns the currently defined direction for the pulse width trigger.

**Returned Format**

```
[:TRIGger:ADVanced:VIOLation:PWIDth:DIRection]
{GTHan|LTHan}<NL>
```

# VIOLation:PWIDth:WIDTh

**Command**

```
:TRIGger:ADVanced:VIOLation:PWIDth:WIDTh <width>
```

This command specifies how wide a pulse must be to trigger the oscilloscope.

`<width>` Pulse width, which can range from 1.5 ns to 160 ms.

**Query**

```
:TRIGger:ADVanced:VIOLation:PWIDth:WIDTh?
```

The query returns the currently defined width for the pulse.

**Returned Format**

```
[:TRIGger:ADVanced:VIOLation:PWIDth:WIDTh] <width><NL>
```

Use Setup Violation Mode to find violations of setup and hold times in your circuit.

**Mode**

Use MODE to select Setup, Hold, or both Setup and Hold time triggering.

You can have the oscilloscope trigger on violations of setup time, hold time, or both setup and hold time. To use Setup Violation Type, the oscilloscope needs a clock waveform, used as the reference, and a data waveform for the trigger source.

`Setup Time Mode` When using the Setup Time Mode, a time window is defined where the right edge is the clock edge and the left edge is the selected time before the clock edge. The waveform must stay outside of the thresholds during this time window. If the waveform crosses a threshold within the time window, a violation event occurs and the oscilloscope triggers.

`Hold Time Mode` When using Hold Time Mode, the waveform must not cross the threshold voltages after the specified clock edge for at least the hold time you have selected. Otherwise, a violation event occurs and the oscilloscope triggers.

`Setup and Hold Time Mode` When using the Setup and Hold Time Mode, if the waveform violates either a setup time or hold time, the oscilloscope triggers.

**Data Source**

Use the data source (DSOurce) command to select the channel used as the data, the low-level data threshold, and the high-level data threshold. For data to be considered valid, it must be below the lower threshold or above the upper threshold during the time of interest.

`DSOurce` Use DSOurce to select the channel you want to use for the data source.

`Low Threshold` Use the low threshold (LTHReshold) to set the minimum threshold for your data. Data is valid below this threshold.

`High Threshold` Use the high threshold (HTHReshold) to set the maximum threshold for your data. Data is valid above this threshold.

**Clock Source**

Use the clock source (CSOurce) command to select the clock source, trigger level, and edge polarity for your clock. Before the trigger circuitry looks for a setup or hold time violation, the clock must pass through the voltage level you have set.

`CSOurce` Use CSOurce to select the channel you want to use for the clock source.

`LEVel` Use LEVel to set voltage level on the clock waveform as given in the data book for your logic family.

`RISing or FALLing` Use RISing or FALLing to select the edge of the clock the oscilloscope uses as a reference for the setup or hold time violation trigger.

**Time**

`Setup Time` Use SETup to set the amount of setup time used to test for a violation. The setup time is the amount of time that the data has to be stable and valid prior to a clock edge. The minimum is 1.5 ns; the maximum is 20 ns.

`Hold Time` Use HOLD to set the amount of hold time used to test for a violation. The hold time is the amount of time that the data has to be stable and valid after a clock edge. The minimum is 1.5 ns; the maximum is 20 ns.

`Setup and Hold` Use SHOLd (Setup and Hold) to set the amount of setup and hold time used to test for a violation.

The setup time is the amount of time that the data has to be stable and valid prior to a clock edge. The hold time is the amount of time that the data waveform has to be stable and valid after a clock edge.

The setup time plus hold time equals 20 ns maximum. So, if the setup time is 1.5 ns, the maximum hold time is 18.5 ns.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

**Set the Mode Before Executing Commands**

Before executing the :TRIGger:ADVanced:VIOLation:SETup commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE VIOLation  and
:TRIGger:ADVanced:VIOLation:MODE SETup  and
:TRIGger:ADVanced:VIOLation:SETup:MODE <setup_mode>
```

Where <setup_mode> includes SETup, HOLD, and SHOLd.

To query the oscilloscope for the advanced trigger violation setup mode, enter:

```
:TRIGger:ADVanced:VIOLation:SETup:MODE?
```

# VIOLation:SETup:MODE

Command

```
:TRIGger:ADVanced:VIOLation:SETup:MODE
{SETup|HOLD|SHOLd}
```

`SETup` When using the setup time mode, a time window is defined where the right edge is the clock edge and the left edge is the selected time before the clock edge. The waveform must stay outside of the trigger level thresholds during this time window. If the waveform crosses a threshold during this time window, a violation event occurs and the oscilloscope triggers.

`HOLD` When using the hold time mode, the waveform must not cross the threshold voltages after the specified clock edge for at least the hold time you have selected. Otherwise, a violation event occurs and the oscilloscope triggers.

`SHOLd` When using the setup and hold time mode, if the waveform violates either a setup time or hold time, the oscilloscope triggers. The total time allowed for the sum of setup time plus hold time is 20 ns maximum.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:MODE?
```

The query returns the currently selected trigger setup violation mode.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:MODE]
{SETup|HOLD|SHOLd}<NL>
```

# VIOLation:SETup:SETup:CSOurce

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce
{CHANnel<N>|EXTernal}
```

This command specifies the clock source for the clock used for the trigger setup violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup or hold time violation.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce?
```

The query returns the currently defined clock source for the trigger setup violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce]
{CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:SETup:SETup:CSOurce:LEVel

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:
LEVel {{CHANnel<N>|EXTernal},<level>}
```

This command specifies the level for the clock source used for the trigger setup violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup or hold time violation.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the voltage level for the trigger setup violation clock waveform, and depends on the type of circuitry logic you are using.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:
LEVel? {CHANnel<N>|EXTernal}
```

The query returns the specified clock source level for the trigger setup violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:LEVel
{CHANnel<N>|EXTernal},] <level><NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

## VIOLation:SETup:SETup:CSOurce:EDGE

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:
EDGE {RISing|FALLing}
```

This command specifies the edge for the clock source used for the trigger setup violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup or hold time violation.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:
EDGE?
```

The query returns the currently defined clock source edge for the trigger setup violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:EDGE]
{RISing|FALLing}<NL>
```

# VIOLation:SETup:SETup:DSOurce

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce
{CHANnel<N>|EXTernal}
```

The data source commands specify the data source for the trigger setup violation.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce?
```

The query returns the currently defined data source for the trigger setup violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce]
{CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:SETup:SETup:DSOurce:HTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:
HTHReshold {{CHANnel<N>|EXTernal},<level>}
```

This command specifies the data source for the trigger setup violation, and the high-level data threshold for the selected data source. Data is valid when it is above the high-level data threshold, and when it is below the low-level data threshold.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the data threshold level for the trigger setup violation, and is used with the high and low threshold data source commands.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:
HTHReshold? {CHANnel<N>|EXTernal}
```

The query returns the specified data source for the trigger setup violation, and the high data threshold for the data source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:HTHReshol
d {CHANnel<N>|EXTernal},] <level><NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:SETup:SETup:DSOurce:LTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:
LTHReshold {{CHANnel<N>|EXTernal},<level>}
```

This command specifies the data source for the trigger setup violation, and the low-level data threshold for the selected data source. Data is valid when it is above the high-level data threshold, and when it is below the low-level data threshold.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the data threshold level for the trigger setup violation, and is used with the high and low threshold data source commands.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:
LTHReshold? {CHANnel<N>|EXTernal}
```

The query returns the specified data source for the trigger setup violation, and the low data threshold for the data source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:LTHReshol
d
{CHANnel<N>|EXTernal},] <level><NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:SETup:SETup:TIME

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:TIME <time>
```

This command specifies the amount of setup time used to test for a trigger violation. The setup time is the amount of time that the data must be stable and valid prior to a clock edge.

`<time>` Setup time, in seconds.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:TIME?
```

The query returns the currently defined setup time for the trigger violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SETup:TIME] <time><NL>
```

# VIOLation:SETup:HOLD:CSOurce

Command

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce
{CHANnel<N>|EXTernal}
```

This command specifies the clock source for the clock used for the trigger hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup or hold time violation.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce?
```

The query returns the currently defined clock source for the trigger hold violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce]
{CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:SETup:HOLD:CSOurce:LEVel

Command

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:
LEVel {{CHANnel<N>|EXTernal},<level>}
```

This command specifies the level for the clock source used for the trigger hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup or hold time violation.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other nfiniium Oscilloscope models.

`<level>` A real number for the voltage level for the trigger hold violation clock waveform, and depends on the type of circuitry logic you are using.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:
LEVel? {CHANnel<N>|EXTernal}
```

The query returns the specified clock source level for the trigger hold violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:LEVel
{CHANnel<N>|EXTernal},] <level><NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:SETup:HOLD:CSOurce:EDGE

Command

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:
EDGE {RISing|FALLing}
```

This command specifies the edge for the clock source used for the trigger hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup or hold time violation.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:
EDGE?
```

The query returns the currently defined clock source edge for the trigger hold violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:EDGE]
{RISing|FALLing}<NL>
```

# VIOLation:SETup:HOLD:DSOurce

Command

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce
{CHANnel<N>|EXTernal}
```

The data source commands specify the data source for the trigger hold violation.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce?
```

The query returns the currently defined data source for the trigger hold violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce]
{CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:SETup:HOLD:DSOurce:HTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:
HTHReshold {{CHANnel<N>|EXTernal},<level>}
```

This command specifies the data source for the trigger hold violation, and the high-level data threshold for the selected data source. Data is valid when it is above the high-level data threshold, and when it is below the low-level data threshold.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the data threshold level for the trigger hold violation, and is used with the high and low threshold data source commands.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:
HTHReshold? {CHANnel<N>|EXTernal}
```

The query returns the specified data source for the trigger hold violation, and the high data threshold for the data source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:HTHReshold
{CHANnel<N>|EXTernal},] <level><NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:SETup:HOLD:DSOurce:LTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:
LTHReshold {{CHANnel<N>|EXTernal},<level>}
```

This command specifies the data source for the trigger hold violation, and the low-level data threshold for the selected data source. Data is valid when it is above the high-level data threshold, and when it is below the low-level data threshold.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the data threshold level for the trigger hold violation, and is used with the high and low threshold data source commands.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:
LTHReshold?  {CHANnel<N>|EXTernal}
```

The query returns the specified data source for the trigger hold violation, and the low data threshold for the data source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:LTHReshold
{CHANnel<N>|EXTernal},] <level><NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:SETup:HOLD:TIME

**Command**

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:TIME <time>
```

This command specifies the amount of hold time used to test for a trigger violation. The hold time is the amount of time that the data must be stable and valid after a clock edge.

`<time>` Hold time, in seconds.

**Query**

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:TIME?
```

The query returns the currently defined hold time for the trigger violation.

**Returned Format**

```
[:TRIGger:ADVanced:VIOLation:SETup:HOLD:TIME] <time><NL>
```

# VIOLation:SETup:SHOLd:CSOurce

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:
{CHANnel<N>|EXTernal}
```

This command specifies the clock source for the clock used for the trigger setup and hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup and hold time violation.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce?
```

The query returns the currently defined clock source for the trigger setup and hold violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce]
{CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:SETup:SHOLd:CSOurce:LEVel

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:
LEVel {{CHANnel<N>|EXTernal},<level>}
```

This command specifies the clock source trigger level for the clock used for the trigger setup and hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup and hold time violation.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the voltage level for the trigger setup and hold violation clock waveform, and depends on the type of circuitry logic you are using.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:
LEVel? {CHANnel<N>|EXTernal}
```

The query returns the specified clock source level for the trigger setup and hold violation level for the clock source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:LEVel
{CHANnel<N>|EXTernal},] <level><NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

## VIOLation:SETup:SHOLd:CSOurce:EDGE

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:
EDGE {RISing|FALLing}
```

This command specifies the clock source trigger edge for the clock used for the trigger setup and hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup and hold time violation.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:
EDGE?
```

The query returns the currently defined clock source edge for the trigger setup and hold violation level for the clock source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:EDGE]
{RISing|FALLing}<NL>
```

# VIOLation:SETup:SHOLd:DSOurce

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce
{CHANnel<N>|EXTernal}
```

The data source commands specify the data source for the trigger setup and hold violation.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce?
```

The query returns the currently defined data source for the trigger setup and hold violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce]
{CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:SETup:SHOLd:DSOurce:HTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:
HTHReshold {{CHANnel<N>|EXTernal},<level>}
```

This command specifies the data source for the trigger setup and hold violation, and the high-level data threshold for the selected data source. Data is valid when it is above the high-level data threshold, and when it is below the low-level data threshold.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the data threshold level for the trigger setup and hold violation, and is used with the high and low threshold data source commands.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:
HTHReshold? {CHANnel<N>|EXTernal}
```

The query returns the specified data source for the trigger setup and hold violation, and the high data threshold for the data source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:HTHReshol
d {CHANnel<N>|EXTernal},] <level><NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:SETup:SHOLd:DSOurce:LTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:
LTHReshold {{CHANnel<N>|EXTernal},<level>}
```

This command specifies the data source for the trigger setup and hold violation, and the low-level data threshold for the selected data source. Data is valid when it is above the high-level data threshold, and when it is below the low-level data threshold.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the data threshold level for the trigger setup and hold violation, and is used with the high and low threshold data source commands.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:
LTHReshold?  {CHANnel<N>|EXTernal}
```

The query returns the specified data source for the setup and trigger hold violation, and the low data threshold for the data source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:LTHReshol
d {CHANnel<N>|EXTernal},] <level><NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

## VIOLation:SETup:SHOLd:SetupTIMe (STIMe)

**Command**

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:SetupTIMe
<time>
```

This command specifies the amount of setup time used to test for both a setup and hold trigger violation. The setup time is the amount of time that the data must be stable and valid before a clock edge.

`<time>` Setup time, in seconds.

**Query**

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:SetupTIMe?
```

The query returns the currently defined setup time for the setup and hold trigger violation.

**Returned Format**

```
[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:SetupTIMe]
<time><NL>
```

# VIOLation:SETup:SHOLd:HoldTIMe (HTIMe)

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:HoldTIMe
<time>
```

This command specifies the amount of hold time used to test for both a setup and hold trigger violation. The hold time is the amount of time that the data must be stable and valid after a clock edge.

`<time>` Hold time, in seconds.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLD:HoldTIMe?
```

The query returns the currently defined hold time for the setup and hold trigger violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SHOLD:HoldTIMe]
<time><NL>
```

# Transition Violation Mode

Use Transition Violation Mode to find any edge in your waveform that violates a rise time or fall time specification. Infiniium Oscilloscopes find a transition violation trigger by looking for any pulses in your waveform with rising or falling edges that do not cross two voltage levels in the amount of time you have specified.

The rise time is measured from the time that your waveform crosses the low threshold until it crosses the high threshold. The fall time is measured from the time that the waveform crosses the high threshold until it crosses the low threshold.

`Source` Use Source to select the channel used for a transition violation trigger.

`Low Threshold` Use Low Threshold to set the low voltage threshold.

`High Threshold` Use High Threshold to set the high voltage threshold.

`Type` Use Type to select Rise Time or Fall Time violation.

`Trigger On` Trigger On parameters include > Time and < Time.

`> Time` Use > Time to look for transition violations that are longer than the time specified.

`< Time` Use < Time to look for transition violations that are less than the time specified.

`Time` Use Time to set the amount of time to determine a rise time or fall time violation.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

**Set the Mode Before Executing Commands**

Before executing the :TRIGger:ADVanced:VIOLation:TRANsition commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE VIOLation  and
:TRIGger:ADVanced:VIOLation:MODE TRANsition
```

To query the oscilloscope for the advanced trigger violation mode, enter:

```
:TRIGger:ADVanced:VIOLation:MODE?
```

## VIOLation:TRANsition

**Command**

```
:TRIGger:ADVanced:VIOLation:TRANsition:
{GTHan|LTHan} <time>
```

This command lets you look for transition violations that are greater than or less than the time specified.

`<time>` The time for the trigger violation transition, in seconds.

**Query**

```
:TRIGger:ADVanced:VIOLation:TRANsition:
{GTHan|LTHan}?
```

The query returns the currently defined time for the trigger transition violation.

**Returned Format**

```
[:TRIGger:ADVanced:VIOLation:TRANsition:{GTHan|LTHan}]
<time><NL>
```

# VIOLation:TRANsition:SOURce

Command

```
:TRIGger:ADVanced:VIOLation:TRANsition:SOURce
{CHANnel<N>|EXTernal}
```

The transition source command lets you find any edge in your waveform that violates a rise time or fall time specification. The oscilloscope finds a transition violation trigger by looking for any pulses in your waveform with rising or falling edges that do not cross two voltage levels in the amount of time you have specified.

<N> An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

Query

```
:TRIGger:ADVanced:VIOLation:TRANsition:SOURce?
```

The query returns the currently defined transition source for the trigger transition violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:TRANsition:SOURce]
{CHANnel<N>|EXTernal}<NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:TRANsition:SOURce:HTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:
HTHReshold {{CHANnel<N>|EXTernal},<level>}
```

This command lets you specify the source and high threshold for the trigger violation transition. The oscilloscope finds a transition violation trigger by looking for any pulses in your waveform with rising or falling edges that do not cross two voltage levels in the amount of time you have specified.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the voltage threshold level for the trigger transition violation, and is used with the high and low threshold transition source commands.

Query

```
:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:
HTHReshold? {CHANnel<N>|EXTernal}
```

The query returns the specified transition source for the trigger transition high threshold violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:HTHReshold
{CHANnel<N>|EXTernal},] <level><NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

# VIOLation:TRANsition:SOURce:LTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:
LTHReshold {{CHANnel<N>|EXTernal},<level>}
```

This command lets you specify the source and low threshold for the trigger violation transition. The oscilloscope finds a transition violation trigger by looking for any pulses in your waveform with rising or falling edges that do not cross two voltage levels in the amount of time you have specified.

`<N>` An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

`<level>` A real number for the voltage threshold level for the trigger transition violation, and is used with the high and low threshold transition source commands.

Query

```
:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:
LTHReshold? {CHANnel<N>|EXTernal}
```

The query returns the currently defined transition source for the trigger transition low threshold violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:LTHReshold
{CHANnel<N>|EXTernal},] <level><NL>
```

**EXTernal is Only Available in Some Infiniium Oscilloscopes**

EXTernal is only available in 2-channel Infiniium Oscilloscope models (including the 54810A and 54820A).

## VIOLation:TRANsition:TYPE

Command

```
:TRIGger:ADVanced:VIOLation:TRANsition:TYPE
{RISetime|FALLtime}
```

This command lets you select either a rise time or fall time transition violation trigger event.

Query

```
:TRIGger:ADVanced:VIOLation:TRANsition:TYPE?
```

The query returns the currently defined transition type for the trigger transition violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:TRANsition:TYPE]
{RISetime|FALLtime}<NL>
```

# Waveform Commands

# Waveform Commands

The WAVeform subsystem is used to transfer waveform data between a computer and the oscilloscope. It contains commands to set up the waveform transfer and to send or receive waveform records to or from the oscilloscope. These WAVeform commands and queries are implemented in the Infiniium Oscilloscopes:

- BANDpass?
- BYTeorder
- CLIPped?
- COMPlete?
- COUNt?
- COUPling?
- DATA
- FORMat
- POINts?
- PREamble
- SOURce
- TYPE?
- VIEW
- XDISplay?
- XINCrement?
- XORigin?
- XRANge?
- XREFerence?
- XUNits?
- YDISplay?
- YINCrement?
- YORigin?
- YRANge?
- YREFerence?
- YUNits?

**Data Acquisition**

When data is acquired using the DIGitize command, the data is placed in the channel or function memory of the specified source. After the DIGitize command executes, the oscilloscope is stopped. If the oscilloscope is restarted over GPIB or from the front panel, the data acquired with the DIGitize command is overwritten.

You can query the preamble, elements of the preamble, or waveform data while the oscilloscope is running, but the data will reflect only the current acquisition, and subsequent queries will not reflect consistent data. For example, if the oscilloscope is running and you query the X origin, the data is queried in a separate GPIB command, and it is likely that the first point in the data will have a different time than that of the X origin. This is due to data acquisitions that may have occurred between the queries. For this reason, Agilent Technologies does not recommend this mode of operation. Instead, you should use the DIGitize command to stop the oscilloscope so that all subsequent queries will be consistent.

**CAUTION** Function data is volatile and must be read following a DIGgitize command or the data will be lost when the oscilloscope is turned off.

**Waveform Data and Preamble**

The waveform record consists of two parts: the preamble and the waveform data. The waveform data is the actual sampled data acquired for the specified source. The preamble contains the information for interpreting the waveform data, including the number of points acquired, the format of the acquired data, and the type of acquired data. The preamble also contains the X and Y increments, origins, and references for the acquired data.

The values in the preamble are set when you execute the DIGitize command. The preamble values are based on the settings of controls in the ACQuire commands subsystem.

Although you can change preamble values with a GPIB computer, you cannot change the way the data is acquired. Changing the preamble values cannot change the type of data that was actually acquired or the number of points actually acquired.

**CAUTION** You must use extreme caution when changing any waveform preamble values to ensure that the data is still useful. For example, setting the number of points in the preamble to a different value from the actual number of points in the waveform results in inaccurate data.

The waveform data and preamble must be read or sent using the separate commands :WAVeform:DATA and :WAVeform:PREamble.

**Data Conversion**

Data sent from the oscilloscope must be scaled for useful interpretation. The values used to interpret the data are the X and Y origins, X and Y increments, and X and Y references. These values can be read from the waveform preamble.

**Conversion from Data Values to Units**

To convert the waveform data values (essentially A/D counts) to real-world units, such as volts, use the following scaling formulas:

Y-axis Units = (data value - Yreference) x Yincrement + Yorigin
X-axis Units = (data index - Xreference) x Xincrement + Xorigin,
where the data index starts at zero: 0, 1, 2, ..., n-1.

The first data point for the time (X-axis units) must be zero, so the time of the first data point is the X origin.

**Data Format for GPIB Transfer**

There are four types of data formats that you can select using the :WAVeform:FORMat command: ASCii, BYTE, WORD, and LONG. Refer to the FORMat command in this chapter for more information on data formats.

# BANDpass?

**Query**

```
:WAVeform:BANDpass?
```

The :WAVeform:BANDpass? query returns an estimate of the maximum and minimum bandwidth limits of the source waveform. The bandwidth limits are computed as a function of the coupling and the selected filter mode. The cutoff frequencies are derived from the acquisition path and software filtering.

**Returned Format**

```
[:WAVeform:BANDpass]<lower_cutoff>,<upper_cutoff><NL>
```

`<lower_cutoff>` Minimum frequency passed by the acquisition system.

`<upper_cutoff>` Maximum frequency passed by the acquisition system.

**Example**

This example places the estimated maximum and minimum bandwidth limits of the source waveform in the string variable, Bandwidth$, then prints the contents of the variable to the computer's screen.

```
10  DIM Bandwidth$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:BANDPASS?"
30  ENTER 707;Bandwidth$
40  PRINT Bandwidth$
50  END
```

# BYTeorder

Command

```
:WAVeform:BYTeorder {MSBFirst | LSBFirst}
```

The :WAVeform:BYTeorder command selects the order in which bytes are transferred to and from the oscilloscope using WORD and LONG formats. If MSBFirst is selected, the most significant byte is transferred first. Otherwise, the least significant byte is transferred first. The default setting is MSBFirst.

**Example**

This example sets up the oscilloscope to send the most significant byte first during data transmission.

```
10  OUTPUT 707;":WAVEFORM:BYTEORDER MSBFIRST"
20  END
```

Query

```
:WAVeform:BYTeorder?
```

The :WAVeform:BYTeorder? query returns the current setting for the byte order.

Returned Format

```
[:WAVeform:BYTeorder] {MSBFirst | LSBFirst}<NL>
```

**Example**

This example places the current setting for the byte order in the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[10]!Dimension variable
20  OUTPUT 707;":WAVEFORM:BYTEORDER?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

**MSBFirst and LSBFirst**

MSBFirst is for microprocessors like Motorola's, where the most significant byte resides at the lower address. LSBFirst is for microprocessors like Intel's, where the least significant byte resides at the lower address.

## CLIPped?

**Query**

```
:WAVeform:CLIPped?
```

The :WAVeform:CLIPped? query returns a "1" if the currently selected waveform is clipped, and a "0" if the waveform is not clipped.

**Returned Format**

```
[:WAVeform:CLIPped] {1 | 0}<NL>
```

**Example**

This example places the current clipped status of the selected waveform in the string variable, Setting$, then prints the contents of the variable.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:CLIPPED?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

## COMPlete?

Query

```
:WAVeform:COMPlete?
```

The :WAVeform:COMPlete? query returns the percent of time buckets that are complete for the currently selected waveform.

For the NORMal, RAW, and INTerpolate waveform types, the percent complete is the percent of the number of time buckets that have data in them, compared to the memory depth.

For the AVERage waveform type, the percent complete is the number of time buckets that have had the specified number of hits divided by the memory depth. The hits are specified by the :WAVeform:COUNt command.

For the VERSus waveform type, percent complete is the least complete of the X-axis and Y-axis waveforms.

Returned Format

```
[:WAVeform:COMPlete] <criteria><NL>
```

`<criteria>` 0 to 100 percent, rounded down to the closest integer.

**Example**

This example places the current completion criteria in the string variable, Criteria$, then prints the contents of the variable to the computer's screen.

```
10  DIM Criteria$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:COMPLETE?"
30  ENTER 707;Criteria$
40  PRINT Criteria$
50  END
```

# COUNt?

Query

```
:WAVeform:COUNt?
```

The :WAVeform:COUNt? query returns the fewest number of hits in all of the time buckets for the currently selected waveform. For the AVERage waveform type, the count value is the fewest number of hits for all time buckets. This value may be less than or equal to the value specified with the :ACQuire:AVERage:COUNt command.

For the NORMal, RAW, INTerpolate, and VERSus waveform types, the count value returned is one, unless the data contains holes (sample points where no data is acquired). If the data contains holes, zero is returned.

Returned Format

```
[:WAVeform:COUNt] <number><NL>
```

<number> An integer. Values range from 1 to 262144 for NORMal, RAW, or INTerpolate types, and from 1 to 32768 for VERSus type.

**Example**

This example places the current count field value in the string variable, Count$, then prints the contents of the variable to the computer's screen.

```
10  DIM Count$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:COUNT?"
30  ENTER 707;Count$
40  PRINT Count$
50  END
```

# COUPling?

Query

```
:WAVeform:COUPling?
```

The :WAVeform:COUPling? query returns the input coupling of the currently selected source.

Returned Format

```
[:WAVeform:COUPling] {AC | DC | DCFifty | LFReject}<NL>
```

**Example**

This example places the current input coupling of the selected waveform in the string variable, Setting$, then prints the contents of the variable.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:COUPLING?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

**See Also**

The :CHANnel<N>:INPut command sets the coupling for a particular channel. You can use the :WAVeform:SOURce command to set the source for the coupling query.

| Source | Return Value |
|---|---|
| CGRade | Lowest numbered channel that is on. |
| HISTogram | The selected channel or for functions, the lowest numbered channel in the function. |
| CHANnel | The channel number |
| FUNCtion | The lowest numbered channel in the function |
| WMEMory | The coupling value that was loaded into the waveform memory. If channel 1 was loaded, it would be the channel 1 coupling value. |

# DATA

**Command**

```
:WAVeform:DATA <block_data>[,<block_data>]
```

The :WAVeform:DATA command transfers waveform data to the oscilloscope over GPIB and stores the data in a previously specified waveform memory. The waveform memory is specified using the :WAVeform:SOURce command. Only waveform memories may have waveform data sent to them. The format of the data being sent must match the format previously specified by the waveform preamble for the destination memory. Color grade data cannot be stored into a waveform memory.

VERSus data is transferred as two arrays. The first array contains the data on the X axis, and the second array contains the data on the Y axis. The two arrays are transferred one at a time over GPIB in a linear format. The number of points sent in each array is equal to the number in the points portion of the preamble.

The full-scale vertical range of the A/D converter will be returned with the data query. You should use the Y-increment, Y-origin, and Y-reference values to convert the full-scale vertical ranges to voltage values. You should use the Y-range and Y-display values to plot the voltage values. All of these reference values are available from the waveform preamble. Refer to "Conversion from Data Values to Units" earlier in this chapter.

`<block_data>` Binary block data in the # format (as described in HP BASIC Image Specifiers below).

**HP BASIC Image Specifiers**

# is an HP BASIC image specifier that suppresses the automatic output of the EOL sequence following the last output item.

K is an HP BASIC image specifier that outputs a number or string in standard form with no leading or trailing blanks.

W is an HP BASIC image specifier that outputs 16-bit words with the most significant byte first.

**Example**

This example sends 1000 bytes of previously saved data to the oscilloscope from the array, Set.

```
10  OUTPUT 707 USING "#,K";:WAVEFORM:DATA #800001000"
20  OUTPUT 707 USING "W";Set(*)
30  END
```

Query

```
:WAVeform:DATA?
```

The :WAVeform:DATA? query outputs waveform data to the computer over the GPIB Interface. The data is copied from a waveform memory, function, or channel buffer previously specified with the :WAVeform:SOURce command. The returned data is described by the waveform preamble.

Returned Format

```
[:WAVeform:DATA] <block_data>[,<block_data>]<NL>
```

**BASIC Example**

This example places the current waveform data from channel 1 of the array Wdata in the word format.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:SOURCE CHANNEL1!Select source
30  OUTPUT 707;":WAVEFORM:FORMAT WORD"!Select word format
40  OUTPUT 707;":WAVEFORM:DATA?"
50  ENTER 707 USING "#,1A";Pound_sign$
60  ENTER 707 USING "#,1D";Header_length
70  ENTER 707 USING "#,"&VAL$(Header_length)&"D";Length
80  Length = Length/2!Length in words
90  ALLOCATE INTEGER Wdata(1:Length)
100 ENTER 707 USING "#,W";Wdata(*)
110 ENTER 707 USING "-K,B";End$
120 END
```

**HP BASIC Image Specifiers**

# is an HP BASIC image specifier that terminates the statement when the last ENTER item is terminated. EOI and line feed are the item terminators.

1A is an HP BASIC image specifier that places the next character received in a string variable.

1D is an HP BASIC image specifier that places the next character in a numeric variable.

W is an HP BASIC image specifier that places the data in the array in word format with the first byte entered as the most significant byte.

-K is an HP BASIC image specifier that places the block data in a string, including carriage returns and line feeds until EOI is true or when the dimensioned length of the string is reached.

B is an HP BASIC specifier that enters the next byte in a variable.

The format of the waveform data must match the format previously specified by the :WAVeform:FORMat, :WAVeform:BYTeorder, and :WAVeform:PREamble commands.

**C Example**

The following example shows how to transfer both BYTE and WORD formatted waveform data to a computer. There is a file on the Infiniium Oscilloscope Example Programs disk called readdata.c in the c directory that contains this program.

```
/* readdata. c */

/* Reading Byte and Word format Example.  This program demonstrates the order
   of commands suggested for operation of the Infiniium oscilloscope via GPIB.
   This program initializes the scope, acquires data, transfers data in both
   the BYTE and WORD formats, converts the data into voltage and time values,
   and stores the data on the PC as time, word voltage values, and byte
   voltage values in a comma-separated file format.  This format is useful
   for spreadsheet applications.  It assumes a SICL GPIB interface card exists
   as 'hpib7' and an Infiniium oscilloscope at address 7.  It also requires a
   waveform connected to Channel 1.
*/

#include <stdio.h>   /* location of: printf() */
#include <stdlib.h>  /* location of: atof(), atoi() */
#include <string.h>  /* location of: strlen() */
#include <sicl.h>

/* Prototypes */
int InitIO( void );
void WriteIO( char *buffer );
unsigned long ReadByte( char *buffer, unsigned long BytesToRead);
unsigned long ReadWord( short *buffer, unsigned long BytesToRead);
void ReadDouble( double *buffer );
void CloseIO( void );
void AcquireData( void );
void GetVoltageConversionFactors( double *yInc,
                                  double *yOrg,
                                  double *yRef );
void GetTimeConversionFactors( double *xInc,
                               double *xOrg,
                               double *xRef );
void CreateTimeData( double xInc,
                     double xOrg,
                     double xRef,
                     unsigned long AcquiredLength,
                     double *TimeValues );
void ConvertWordDataToVolts( short *byteData,
                             double *byteVolts,
                             unsigned long AcquiredLength,
                             double yInc,
                             double yOrg,
                             double yRef );
```

```
void ConvertByteDataToVolts( char *byteData,
                             double *byteVolts,
                             unsigned long AcquiredLength,
                             double yInc,
                             double yOrg,
                             double yRef );
void WriteCsvToFile( double *TimeValues,
                     double *wordVolts,
                     double *byteVolts,
                     unsigned long AcquiredLength );
unsigned long SetupDataTransfer( void );

/* Defines */
#define MAX_LENGTH 131072
#define INTERFACE "hpib7"
#define DEVICE_ADDR "hpib7,7"
#define TRUE 1
#define FALSE 0
#define IO_TIMEOUT 20000

/* Globals */
INST bus;
INST scope;
double TimeValues[MAX_LENGTH]; /* Time value of data */
double byteVolts[MAX_LENGTH];  /* Voltage value of data in byte format */
double wordVolts[MAX_LENGTH];  /* Voltage value of data in word format */
short wordData[MAX_LENGTH/2];  /* Buffer for reading word format data */
char byteData[MAX_LENGTH];     /* Buffer for reading byte format data */
```

```
void main( void )
{
   double xOrg=0L, xRef=0L, xInc=0L; /* Values used to create time data */
   double yOrg=0L, yRef=0L, yInc=0L; /* Values used to convert data to volts */
   char Term;
   unsigned long BytesToRead;

   if ( !InitIO() ) {
      exit( 1 );
   }

   AcquireData();

   WriteIO( ":WAVeform:FORMat WORD" );  /* Setup transfer format */
   WriteIO( ":WAVeform:BYTeorder LSBFirst" );/* Setup transfer of LSB first */
   WriteIO( ":WAVeform:SOURce CHANnel1" ); /* Waveform data source channel 1 */

   GetVoltageConversionFactors( &yInc, &yOrg, &yRef );
   BytesToRead = SetupDataTransfer();
   ReadWord( wordData, BytesToRead );
   ReadByte( &Term, 1L ); /* Read termination character */
   ConvertWordDataToVolts( wordData, wordVolts, BytesToRead,
                           yInc, yOrg, yRef );

   WriteIO(":WAVeform:FORMat BYTE");/* Setup transfer format */

   GetVoltageConversionFactors( &yInc, &yOrg, &yRef );
   BytesToRead = SetupDataTransfer();
   ReadByte( byteData, BytesToRead );
   ReadByte( &Term, 1L ); /* Read the termination character */
   ConvertByteDataToVolts( byteData, byteVolts, BytesToRead,
                           yInc, yOrg, yRef );

   GetTimeConversionFactors( &xInc, &xOrg, &xRef );
   CreateTimeData( xInc, xOrg, xRef, BytesToRead, TimeValues );

   WriteCsvToFile( TimeValues, wordVolts, byteVolts, BytesToRead );

   CloseIO( );
}
```

```
/*****************************************************************************
* Function name:  InitIO
*   Parameters:   none
*   Return value: none
*   Description:  This routine initializes the SICL environment.  It sets up
*                 error  handling, opens both an interface and device session,
*                 sets timeout values, clears the GPIB interface card,
*                 and clears the oscilloscope's GPIB card by performing a
*                 Selected Device Clear.
*****************************************************************************/

int InitIO( void )
{

   ionerror( I_ERROR_EXIT );   /* set-up interface error handling */

   bus = iopen( INTERFACE );   /* open interface session */
   if ( bus == 0 ) {
      printf( "Bus session invalid\n" );
      return FALSE;
   }

   itimeout( bus, IO_TIMEOUT );       /* set bus timeout */
   iclear( bus );                     /* clear the interface */

   scope = iopen( DEVICE_ADDR );      /* open the scope device session */
   if ( scope == 0) {
      printf( "Scope session invalid\n" );
      iclose( bus );     /* close interface session */
      _siclcleanup();    /* required for 16-bit applications */
      return FALSE;
   }

   itimeout( scope, IO_TIMEOUT );     /* set device timeout */
   iclear( scope );     /* perform Selected Device Clear on oscilloscope */

   return TRUE;
}
```

```
/*****************************************************************************
* Function name:  WriteIO
*   Parameters:   char *buffer which is a pointer to the character
*                   string to be output
*   Return value: none
*   Description:  This routine outputs strings to the oscilloscope device
*                 session using SICL commands.
*****************************************************************************/

void WriteIO( char *buffer )
{
   unsigned long actualcnt;
   unsigned long BytesToRead;
   int send_end = 1;

   BytesToRead = strlen( buffer );

   iwrite( scope, buffer, BytesToRead, send_end, &actualcnt );

}

/*****************************************************************************
* Function name:  ReadByte
*   Parameters:   char *buffer which is a pointer to the array to store
*                   the read bytes
*                 unsigned long BytesToRead which indicates the maximum
*                   number of bytes to read
*   Return value: integer which indicates the actual number of bytes read
*   Description:  This routine inputs strings from the scope device session
*                 using SICL commands.
*****************************************************************************/

unsigned long ReadByte( char *buffer, unsigned long BytesToRead )
{
   unsigned long BytesRead;
   int reason;

   BytesRead = BytesToRead;
   iread( scope, buffer, BytesToRead, &reason, &BytesRead );

   return BytesRead;
}
```

```
/*****************************************************************************
* Function name:  ReadWord
*   Parameters:   short *buffer which is a pointer to the word array
*                   to store the bytes read
*                 unsigned long BytesToRead which indicates the maximum
*                   number of bytes to read
*   Return value: integer which indicates the actual number of bytes read
*   Description:  This routine inputs an array of short values from the
*                 oscilloscope device session using SICL commands.
*****************************************************************************/

unsigned long ReadWord( short *buffer, unsigned long BytesToRead )
{
   long BytesRead;
   int reason;

   BytesRead = BytesToRead;
   iread( scope, (char *) buffer, BytesToRead, &reason, &BytesRead );

   return BytesRead;
}

/*****************************************************************************
* Function name:  ReadDouble
*   Parameters:   double *buffer which is a pointer to the float value to read
*   Return value: none
*   Description:  This routine inputs a float value from the oscilloscope
*   device session using SICL commands.
*****************************************************************************/

void ReadDouble( double *buffer )
{
   iscanf( scope, "%lf", buffer );
}
```

```
/*****************************************************************************
*  Function name: close_IO
*   Parameters:   none
*   Return value: none
*   Description:  This routine closes device and interface sessions for the
*                 SICL environment, and calls the routine _siclcleanup
*                 which de-allocates resources used by the SICL environment.
*****************************************************************************/

void CloseIO( void )
{

   iclose( scope );    /* close device session */
   iclose( bus );      /* close interface session */

   _siclcleanup();     /* required for 16-bit applications */

}

/*****************************************************************************
* Function name:  AcquireData
*   Parameters:   none
*   Return value: none
*   Description:   This routine acquires data using the current
*                  oscilloscope settings.
*****************************************************************************/

void AcquireData( void )
{
   /*
    * The root level :DIGitize command is recommended for acquiring new
    * waveform data. It initialize's the oscilloscope's data buffers,
    * acquires new data, and ensures that acquisition criteria are met
    * before the acquisition is stopped.  Note that the display is
    * automatically turned off when you use this form of the :DIGitize
    * command and must be turned on to view the captured data on screen.
    */

   WriteIO(":DIGitize CHANnel1");
   WriteIO(":CHANnel1:DISPlay ON");

}
```

```
/*****************************************************************************
* Function name:  GetVoltageConversionFactors
*   Parameters:   double yInc which is the voltage difference represented by
*                   adjacent waveform data digital codes.
*                 double yOrg which is the voltage value of digital code 0.
*                 double yRef which is the reference point for yOrg.
*   Return value: none
*   Description:  This routine reads the conversion factors used to convert
*                 waveform data to volts.
*****************************************************************************/

void GetVoltageConversionFactors( double *yInc, double *yOrg, double *yRef )
{

   /* Read values which are used to convert data to voltage values */

   WriteIO(":WAVeform:YINCrement?");
   ReadDouble( yInc );

   WriteIO(":WAVeform:YORigin?");
   ReadDouble( yOrg );

   WriteIO(":WAVeform:YREFerence?");
   ReadDouble( yRef );

}
```

```
/*****************************************************************************
* Function name:  SetupDataTransfer
*   Parameters:   none
*   Return value: Number of bytes of waveform data to read.
*   Description: This routine sets up the waveform data transfer and gets
*                the number of bytes to be read.
*****************************************************************************/

unsigned long SetupDataTransfer( void )
{
   unsigned long BytesToRead;
   char header_str[8];
   char cData;
   unsigned long BytesRead;

   WriteIO( ":WAVeform:DATA?" );     /* Request waveform data */

   /* Find the # character */

   do {
      ReadByte( &cData, 1L );
   } while ( cData != '#' );

   /* Read the next byte which tells how many bytes to read for the number
    * of waveform data bytes to transfer value.
    */

   ReadByte( &cData, 1L );
   BytesToRead = cData - '0';  /* Convert to a number */

   /* Reads the number of data bytes that will be transfered */

   BytesRead = ReadByte( header_str, BytesToRead );
   header_str[BytesRead] = '\0';
   BytesToRead = atoi( header_str );

   return BytesToRead;

}
```

```
/*****************************************************************************
* Function name:  GetTimeConversionFactors
*   Parameters:   double xInc which is the time between consecutive
*                   sample points.
*                 double xOrg which is the time value of the first data point.
*                 double xRef which is the reference point for xOrg.
*   Return value: none
*   Description:  This routine transfers the waveform conversion
*                 factors for the time values.
*****************************************************************************/

void GetTimeConversionFactors( double *xInc, double *xOrg, double *xRef )
{

   /* Read values which are used to create time values */

   WriteIO(":WAVeform:XINCrement?");
   ReadDouble( xInc );

   WriteIO(":WAVeform:XORigin?");
   ReadDouble( xOrg );

   WriteIO(":WAVeform:XREFerence?");
   ReadDouble( xRef );

}
```

```
/*****************************************************************************
* Function name:  CreateTimeData
*   Parameters:   double xInc which is the time between consecutive
*                   sample points
*                 double xOrg which is the time value of the first data point
*                 double xRef which is the reference point for xOrg
*                 unsigned long AcquiredLength which is the number of
*                   data points
*                 double TimeValues is a pointer to the array where time
*                   values are stored
*   Return value: none
*   Description:  This routine converts the data to time values using
*                 the values that describe the waveform.  These values are
*                 stored in global variables.
*****************************************************************************/

void CreateTimeData( double xInc, double xOrg, double xRef,
                     unsigned long AcquiredLength, double *TimeValues )
{
   unsigned long i;

   for (i = 0; i < AcquiredLength; i++) {
      TimeValues[i] =((i - xRef) * xInc) + xOrg; /* calculate time values */
   }

}
```

```
/*****************************************************************************
* Function name:  ConvertWordDataToVolts
*   Parameters:   short *wordData which is a pointer to the array
*                   of read word values
*                 double *wordVolts which is a pointer to the array of
*                   calculated voltages
*                 unsigned long AcquiredLength which is the number of data
*                   bytes read
*                 double yInc which is the voltage difference represented
*                   by adjacent waveform data digital codes.
*                 double yOrg which is the voltage value of digital code 0.
*                 double yRef which is the reference point for yOrg.
*   Return value: none
*   Description: This routine converts the word format waveform data to
*                voltage values using values that describe the waveform.
*                These values are stored in global arrays for use by
*                other routines.
*****************************************************************************/

void ConvertWordDataToVolts( short *wordData, double *wordVolts,
                             unsigned long AcquiredLength,
                             double yInc, double yOrg, double yRef )
{
   unsigned long i;

   for (i = 0; i < AcquiredLength/2; i++) {
      /* calculate voltage values */
      wordVolts[i] = ((wordData[i] - yRef) * yInc) + yOrg;
   }

}
```

```
/*****************************************************************************
* Function name:  ConvertByteDataToVolts
*   Parameters:   short *byteData which is a pointer to the array of
*                   read byte values
*                 double *byteVolts which is a pointer to the array of
*                   calculated voltages
*                 unsigned long AcquiredLength which is the number of data
*                   bytes read
*                 double yInc which is the voltage difference represented
*                   by adjacent waveform data digital codes.
*                 double yOrg which is the voltage value of digital code 0.
*                    double yRef which is the reference point for yOrg.
*   Return value: none
*   Description:  This routine converts the byte format waveform data to
*                 voltage values using the values that describe the
*                 waveform.  These values are stored in global variables.
*****************************************************************************/

void ConvertByteDataToVolts( char *byteData, double *byteVolts,
                             unsigned long AcquiredLength,
                             double yInc, double yOrg, double yRef )
{
  unsigned long i;

  for (i = 0; i < AcquiredLength; i++) {
    /* calculate voltage values */
    byteVolts[i] = ((byteData[i] - yRef) * yInc) + yOrg;
  }

}
```

```
/*****************************************************************************
* Function name:  WriteCsvToFile
*   Parameters:   double *TimeValues which is a pointer to an array of
*                   calculated time values
*                 double *wordVolts which is a pointer to an array of
*                   calculated word format voltage values
*                 double *byteVolts which is a pointer to an array of
*                   calculated byte format voltage values
*                 unsigned long AcquiredLength which is the number of data
*                   points read
*   Return value: none
*   Description:  This routine stores the time and voltage information about
*                 the waveform as time, word format voltage, and byte format
*                 voltage separated by commas to a file.
*****************************************************************************/

void WriteCsvToFile( double *TimeValues, double *wordVolts,
                     double *byteVolts, unsigned long AcquiredLength )
{
  FILE *fp;
  unsigned long i;

  fp = fopen( "pairs.csv", "wb" ); /* Open file in binary mode - clear file
                                      if it already exists */

  if (fp != NULL) {

     fprintf( fp, "Time,Word Volts,Byte Volts\n");

     for ( i = 0; i < AcquiredLength; i++ ) {
        fprintf( fp, "%e,%f,%f\n", TimeValues[i], wordVolts[i], byteVolts[i] );
     }

     fclose( fp );
  }
  else {
     printf("Unable to open file 'pairs.csv'\n");
  }

}
```

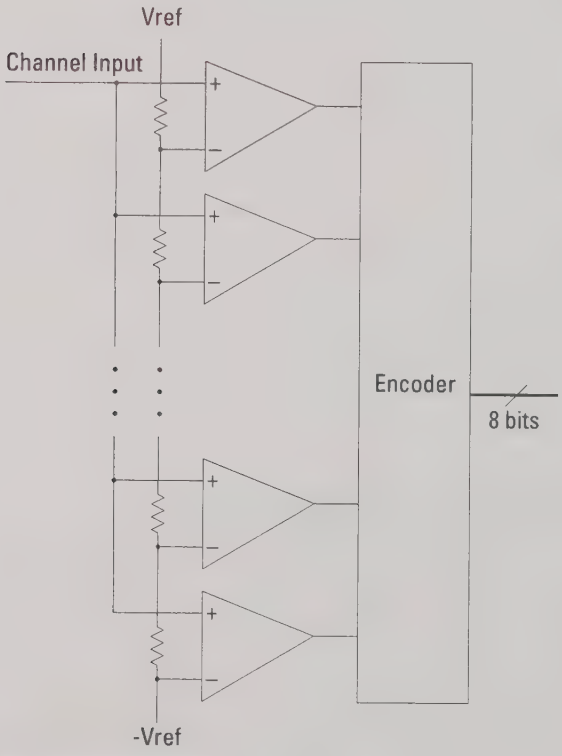
### Understanding WORD and BYTE Formats

Before you can understand how the WORD and BYTE downloads work, it is necessary to understand how Infiniium creates waveform data.

Analog-to-digital Conversion Basics

The input channel of every digital sampling oscilloscope contains an analog-to-digital converter (ADC) as shown in Figure 25-1. The 8-bit ADC in Infiniium consists of 256 voltage comparators. Each comparator has two inputs. One input is connected to a reference dc voltage level and the other input is connected to the channel input. When the voltage of the waveform on the channel input is greater than the dc level, then the comparator output is a 1 otherwise the output is a 0. Each of the comparators has a different reference dc voltage. The output of the comparators is converted into an 8-bit integer by the encoder.

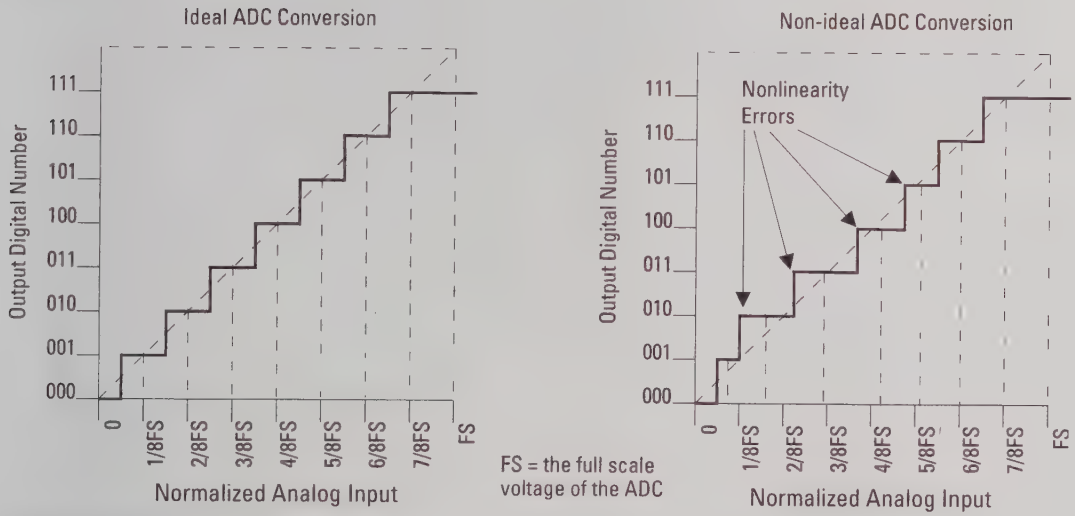
**Figure 25-1**




**Block Diagram of an ADC**

All ADCs have non-linearity errors which, if not corrected, can give less accurate vertical measurement results. For example, the non-linearity error for a 3-bit ADC is shown in the following figure.

**Figure 25-2**

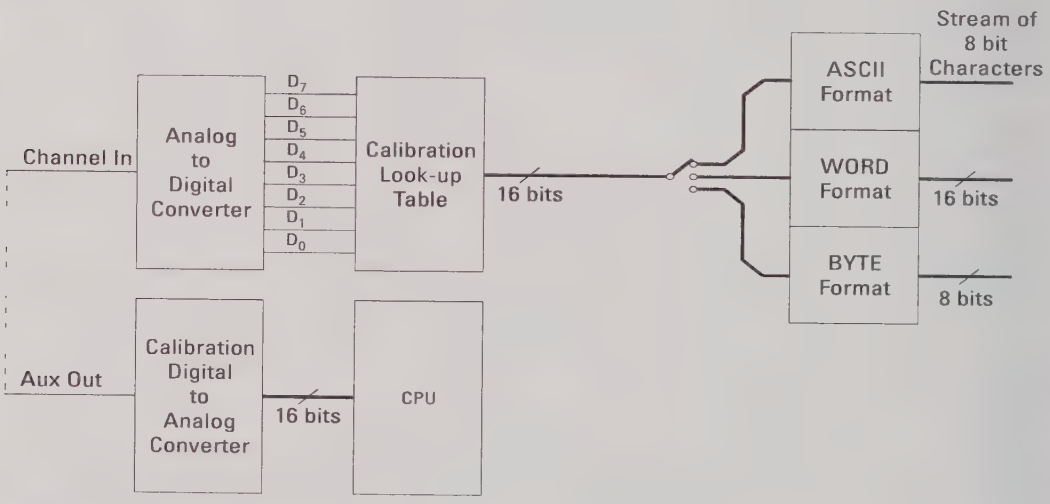



**ADC Non-linearity Errors for a 3-bit ADC**

The graph on the left shows an ADC which has no non-linearity errors. All of the voltage levels are evenly spaced producing output codes that represent evenly spaced voltages. In the graph on the right, the voltages are not evenly spaced with some being wider and some being narrower than the others.

When you calibrate your Infiniium, the input to each channel, in turn, is connected to the Aux Out connector. The Aux Out is connected to a 16-bit digital-to-analog convertor (DAC) whose input is controlled by Infiniium's CPU. There are 65,536 dc voltage levels that are produced by the 16-bit DAC at the Aux Out. At each dc voltage value, the output of the ADC is checked to see if a new digital code is produced. When this happens, a 16-bit correction factor is calculated for that digital code and this correction factor is stored in a Calibration Look-up Table.

**Figure 25-3**




**Data Flow in Infiniium**

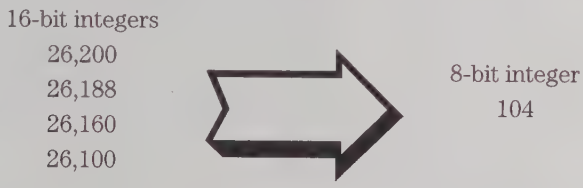
This process continues until all 256 digital codes are calibrated. The calibration process removes most of the non-linearity error of the ADC which yields more accurate vertical voltage values.

During normal operation of the oscilloscope, the output of the ADC is used as an address to the Calibration Look-up Table which produces 16-bit data for the oscilloscope to process and display. The output of the ADC is a signed 8-bit integer and the output of the Calibration Look-up Table is a signed 16-bit integer. If the amplitude of the input waveform is larger than the maximum dc reference level of the ADC, the ADC will output the maximum 8-bit value that it can (255). This condition is called ADC clipping. When the 255 digital code is applied to the Calibration Look-up Table, a 16-bit value, such as 26,188 could be produced which represents an ADC clipped value. This number will vary from one oscilloscope to the next.

**WORD and BYTE Data Formats**

When downloading the waveform data in WORD format, the 16-bit signed integer value for each data point is sent in two consecutive 8-bit bytes over GPIB. Whether the least significant byte (LSB) or the most significant byte (MSB) is sent first depends on the byte order determined by the BYTeorder command.

Before downloading the waveform data in BYTE format, each 16-bit signed integer is converted into an 8-bit signed integer. Because there are more possible 16-bit integers than there are 8-bit integers, a range of 16-bit integers is converted into single 8-bit numbers. For example, the following 16-bit numbers are all converted into one 8-bit number.




This conversion is what makes the BYTE download format less accurate than the WORD format.

# FORMat

Command

```
:WAVeform:FORMat {ASCii | BYTE | LONG | WORD}
```

The :WAVeform:FORMat command sets the data transmission mode for waveform data output. This command controls how the data is formatted when it is sent from the oscilloscope, and pertains to all waveforms. The default format is ASCii.

**Selecting a Format**

| Type | Advantages | Disadvantages |
|---|---|---|
| ASCii | Data is returned as voltage values and does not need to be converted and is as accurate as WORD format. | Very slow data download rate. |
| BYTE | Data download rate is twice as fast as the WORD format. | Data is less accurate than the WORD format. |
| WORD | Data is the most accurate. | Data download rate takes twice as long as the BYTE format |
| LONG | This format is only used to download HISTogram sources. | Cannot be used to download WMEMory, FUNCtion, or CHANnel sources. |

`ASCii` ASCii-formatted data consists of waveform data values converted to the currently selected units, such as volts, and are output as a string of ASCII characters with each value separated from the next value by a comma. The values are formatted in floating point engineering notation. For example:

8.0836E+2,8.1090E+2,...,-3.1245E-3

The ASCii format does not send out the header information indicating the number of bytes being downloaded.

In ASCii format:

- The value "99.999E+36" represents a hole value. A hole can occur when you are using the equivalent time sampling mode when during a single acquisition not all of the acquisition memory locations contain sampled waveform data. It can take several acquisitions in the equivalent time sampling mode to fill all of the memory locations.

`BYTE` BYTE-formatted data is formatted as signed 8-bit integers. If you use BASIC, you need to create a function to convert these signed bits to signed integers. In BYTE format:

- The value 125 represents a hole value. A hole can occur when you are using the equivalent time sampling mode when during a single acquisition not all of the acquisition memory locations contain sampled waveform data. It can take several acquisitions in the equivalent time sampling mode to fill all of the memory locations.

The waveform data values are converted from 16-bit integers to 8-bit integers before being downloaded to the computer. For more information see "Understanding WORD and BYTE Formats" on page 25-28.

`WORD` WORD-formatted data is transferred as signed 16-bit integers in two bytes. If :WAVeform:BYTeorder is set to MSBFirst, the most significant byte of each word is sent first. If the BYTeorder is LSBFirst, the least significant byte of each word is sent first. In WORD format:

- The value 31232 represents a hole level. A hole can occur when you are using the equivalent time sampling mode when during a single acquisition not all of the acquisition memory locations contain sampled waveform data. It can take several acquisitions in the equivalent time sampling mode to fill all of the memory locations.

For more information see "Understanding WORD and BYTE Formats" on page 25-28.

`LONG` LONG-formatted data can only be used when the SOURce is set to HISTogram and is transferred as signed 32-bit integers in four bytes. If :WAVeform:BYTeorder is set to MSBFirst, the most significant byte of each long word is sent first. If the BYTeorder is LSBFirst, the least significant byte of each long word is sent first. In LONG format:

- The value 2046820352 represents a hole level. A hole can occur when you are using the equivalent time sampling mode when during a single acquisition not all of the acquisition memory locations contain sampled waveform data. It can take several acquisitions in the equivalent time sampling mode to fill all of the memory locations.

**Example**

This example selects the WORD format for waveform data transmission.

```
10  OUTPUT 707;":WAVEFORM:FORMAT WORD"
20  END
```

Query

```
:WAVeform:FORMat?
```

The :WAVeform:FORMat? query returns the current output format for transferring waveform data.

Returned Format

```
[:WAVeform:FORMat] {ASCii | BYTE | LONG | WORD}<NL>
```

**Example**

This example places the current output format for data transmission in the string variable, Mode$, then prints the contents of the variable to the computer's screen.

```
10  DIM Mode$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:FORMAT?"
30  ENTER 707;Mode$
40  PRINT Mode$
50  END
```

# POINts?

**Query**

```
:WAVeform:POINts?
```

The :WAVeform:POINts? query returns the points value in the current waveform preamble. The points value is the number of time buckets contained in the waveform selected with the :WAVeform:SOURce command.

**Returned Format**

```
[:WAVeform:POINts] <points><NL>
```

<points> An integer. Values range from 1 to 262144. See the :ACQuire:POINts command for more information.

**Example**

This example places the current acquisition length in the numeric variable, Length, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:POINTS?"
30  ENTER 707;Length
40  PRINT Length
50  END
```

**Turn Headers Off**

When you are receiving numeric data into numeric variables, you should turn the headers off. Otherwise, the headers may cause misinterpretation of returned data.

**See Also**

The :ACQuire:POINts command in the ACQuire Commands chapter.

# PREamble

Command

```
:WAVeform:PREamble <preamble_data>
```

The :WAVeform:PREamble command sends a waveform preamble to the previously selected waveform memory in the oscilloscope. The preamble contains the scaling and other values used to describe the data. The waveform memory is specified with the :WAVeform:SOURce command. Only waveform memories may have waveform data sent to them.

The preamble can be used to translate raw data into time and voltage values. The following lists the elements in the preamble.

`<preamble_data>` <format>, <type>, <points>, <count> ,
<X increment>, <X origin>, < X reference>,
<Y increment>, <Y origin>, <Y reference>,
<coupling>,
<X display range>, <X display origin>,
<Y display range>, <Y display origin>,
<date>, <time>,
<frame model #>,
<acquisition mode>, <completion>,
<X units>, <Y units>,
<max bandwidth limit>, <min bandwidth limit>

`<date>` A string containing the date in the format DD MMM YYYY, where DD is the day, 1 to 31; MMM is the month; and YYYY is the year.

`<time>` A string containing the time in the format HH:MM:SS:TT, where HH is the hour, 0 to 23, MM is the minute, 0 to 59, SS is the second, 0 to 59, and TT is the hundreds of seconds, 0 to 99.

`<frame_model_#>` A string containing the model number and serial number of the frame in the format MODEL#:SERIAL#.

`<format>` 0 for ASCii format.
1 for BYTE format.
2 for WORD format.
3 for LONG format.

`<type>` 1 for RAW type.
2 for AVERage type.
3 VHIStogram.
4 HHIStogram.
5 for VERSUS type.
6 for INTERPOLATE type.
7 not used.
8 for CGRade type.
9 not used.
10 PDETect.

`<acquisition _mode>` 0 for RTIMe mode.
1 for ETIMe mode.
2 not used.
3 PDETect.

`<coupling>` 0 for AC coupling.
1 for DC coupling.
2 for DCFIFTY coupling.
3 for LFREJECT coupling.

`<x_units>` 0 for UNKNOWN units.
`<y_units>` 1 for VOLT units.
2 for SECOND units.
3 for CONSTANT units.
4 for AMP units.
5 for DECIBEL units.

See Table 25-1 for descriptions of all the waveform preamble elements.

**HP BASIC Image Specifiers**

# is an HP BASIC image specifier that suppresses the automatic output of the EOL sequence following the last output item.

K is an HP BASIC image specifier that outputs a number or string in standard form with no leading or trailing blanks.

Query

```
:WAVeform:PREamble?
```

The :WAVeform:PREamble? query outputs a waveform preamble to the computer from the waveform source, which can be a waveform memory or channel buffer.

Returned Format

```
[:WAVeform:PREamble] <preamble_data><NL>
```

**Example**

This example outputs the current waveform preamble for the selected source to the string variable, Preamble$.

```
10  DIM Preamble$[250]!Dimension variable
20  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
30  OUTPUT 707;":WAVEFORM:PREAMBLE?"
40  ENTER 707 USING "-K";Preamble$
50  END
```

**Placing the Block in a String**

-K is an HP BASIC image specifier that places the block data in a string, including carriage returns and line feeds, until EOI is true, or when the dimensioned length of the string is reached.

**Table 25-1** **Waveform Preamble Elements**

| Element | Description |
|---|---|
| Format | The format value describes the data transmission mode for waveform data output. This command controls how the data is formatted when it is sent from the oscilloscope. (See :WAVeform:FORMat.) |
| Type | This value describes how the waveform was acquired. (See also the :WAVeform:TYPE? query.) |
| Points | The number of data points or data pairs contained in the waveform data. (See :ACQuire:POINts.) |
| Count | For the AVERAGE waveform type, the count value is the minimum count or fewest number of hits for all time buckets. This value may be less than or equal to the value requested with the :ACQuire:AVERage:COUNt command. For NORMAL, RAW, INTERPOLATE, and VERSUS waveform types, this value is 0 or 1. The count value is ignored when it is sent to the oscilloscope in the preamble. (See :WAVeform:TYPE and :ACQuire:COUNt.) |
| X Increment | The X increment is the duration between data points on the X axis. For time domain waveforms, this is the time between points. (See the :WAVeform:XINCrement? query.) |
| X Origin | The X origin is the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. This value is treated as a double precision 64-bit floating point number. (See the :WAVeform:XORigin? query.) |
| X Reference | The X reference is the data point associated with the X origin. It is at this data point that the X origin is defined. In this oscilloscope, the value is always zero. (See the :WAVeform:XREFerence? query.) |
| Y Increment | The Y increment is the duration between Y-axis levels. For voltage waveforms, it is the voltage corresponding to one level. (See the :WAVeform:YINCrement? query.) |
| Y Origin | The Y origin is the Y-axis value at level zero. For voltage waveforms, it is the voltage at level zero. (See the :WAVeform:YORigin? query.) |
| Y Reference | The Y reference is the level associated with the Y origin. It is at this level that the Y origin is defined. In this oscilloscope, this value is always zero. (See the :WAVeform:YREFerence? query.) |
| Coupling | The input coupling of the waveform. The coupling value is ignored when sent to the oscilloscope in the preamble. (See the :WAVeform:COUPling? query.) |
| X Display Range | The X display range is the X-axis duration of the waveform that is displayed. For time domain waveforms, it is the duration of time across the display. (See the :WAVeform:XRANge? query.) |
| X Display Origin | The X display origin is the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. This value is treated as a double precision 64-bit floating point number. (See the :WAVeform:XDISplay? query.) |

| Element | Description |
|---|---|
| Y Display Range | The Y display range is the Y-axis duration of the waveform which is displayed. For voltage waveforms, it is the amount of voltage across the display. (See the :WAVeform:YRANge? query.) |
| Y Display Origin | The Y-display origin is the Y-axis value at the center of the display. For voltage waveforms, it is the voltage at the center of the display. (See the :WAVeform:YDISplay? query.) |
| Date | The date that the waveform was acquired or created. |
| Time | The time that the waveform was acquired or created. |
| Frame Model # | The model number of the frame that acquired or created this waveform. The frame model number is ignored when it is sent to an oscilloscope in the preamble. |
| Acquisition Mode | The acquisition sampling mode of the waveform. (See :ACQuire:MODE.) |
| Complete | The complete value is the percent of time buckets that are complete. The complete value is ignored when it is sent to the oscilloscope in the preamble. (See the :WAVeform:COMPlete? query.) |
| X Units | The X-axis units of the waveform. (See the :WAVeform:XUNits? query.) |
| Y Units | The Y-axis units of the waveform. (See the :WAVeform:YUNits? query.) |
| Band Pass | The band pass consists of two values that are an estimation of the maximum and minimum bandwidth limits of the source waveform. The bandwidth limit is computed as a function of the selected coupling and filter mode. (See the :WAVeform:BANDpass? query.) |

**See Also**

:WAVeform:DATA

## SOURce

Command

```
:WAVeform:SOURce {WMEMory<N> | FUNCtion<N> |
CHANnel<N> | HISTogram}
```

The :WAVeform:SOURce command selects a channel, function, waveform memory, or histogram as the waveform source.

<N> CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

Integers, 1 - 4, representing the selected function or waveform memory.

**Example**

This example selects channel 1 as the waveform source.

```
10  OUTPUT 707;":WAVEFORM:SOURCE CHANNEL1"
20  END
```

Query

```
:WAVeform:SOURce?
```

The :WAVeform:SOURce? query returns the currently selected waveform source.

Returned Format

```
[:WAVeform:SOURce] {WMEMory<N> | FUNCtion<N> | CHANnel<N>
|HISTogram}<NL>
```

**Example**

This example places the current selection for the waveform source in the string variable, Selection$, then prints the contents of the variable to the computer's screen.

```
10  DIM Selection$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:SOURCE?"
30  ENTER 707;Selection$
40  PRINT Selection$
50  END
```

# TYPE?

Query

```
:WAVeform:TYPE?
```

The :WAVeform:TYPE? query returns the current acquisition data type for the currently selected source. The type returned describes how the waveform was acquired. The waveform type may be RAW, INTerpolate, AVERage, COLorgrade, HHIStogram, VHISTogram, or VERSus.

`RAW` RAW data consists of one data point in each time bucket with no interpolation.

`INTerpolate` In the INTerpolate acquisition type, the last data point in each time bucket is stored, and additional data points between the acquired data points are filled by interpolation.

`AVERage` AVERage data consists of the average of the first $n$ hits in a time bucket, where $n$ is the value in the count portion of the preamble. Time buckets that have fewer than $n$ hits return the average of the data they contain. If the :ACQuire:COMPlete parameter is set to 100%, then each time bucket must contain the number of data hits specified with the :ACQuire:AVERage:COUNt command.

`VERSus` VERSus data consists of two arrays of data: one containing the X-axis values, and the other containing the Y-axis values. Versus waveforms can be generated using the FUNCtion subsystem commands.

`COLorgrade` The color grade database is transferred using unsigned values in the word format. The database is transferred as a block of data representing a two-dimensional array of 256 rows by 451 columns. The database may be generated using the DISplay:CGRade command.

`HHIStogram` The data is a horizontal histogram. Histograms are transferred using the LONG format. They can be generated using the Histogram subsystem commands.

`VHIStogram` The data is a vertical histogram. Histograms are transferred using the LONG format. They can be generated using the Histogram subsystem commands.

Returned Format

```
[:WAVeform:TYPE] {RAW | INTerpolate | AVERage |
VERSus | COLorgrade | HHIStogram | VHIStogram}<NL>
```

**Example**

This example places the current acquisition data type in the string variable, Type$, then prints the contents of the variable to the computer's screen.

```
10  DIM Type$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:TYPE?"
30  ENTER 707;Type$
40  PRINT Type$
50  END
```

# VIEW

Command

```
:WAVeform:VIEW {ALL | MAIN | WINDow}
```

The :WAVeform:VIEW command selects which view of the waveform is selected for data and preamble queries. You can set the command to ALL, MAIN, or WINDow. The view has different meanings depending upon the waveform source selected. The default setting for this command is ALL.

`Channels` For channels, you may select ALL, MAIN, or WINDow views. If you select ALL, all of the data in the waveform record is referenced. If you select MAIN, only the data in the main time base range is referenced. The first value corresponds to the first time bucket in the main time base range, and the last value corresponds to the last time bucket in the main time base range. If WINDow is selected, only data in the delayed view is referenced. The first value corresponds to the first time bucket in the delayed view and the last value corresponds to the last time bucket in the delayed view.

`Memories` For memories, if you specify ALL, all the data in the waveform record is referenced. WINDow and MAIN refer to the data contained in the memory time base range for the particular memory. The first value corresponds to the first time bucket in the memory time base range, and the last value corresponds to the last time bucket in the memory time base range.

`Functions` For functions, ALL, MAIN, and WINDow refer to all of the data in the waveform record.

Table 25-2 summarizes the parameters for this command for each source.

**Example**

This example sets up the oscilloscope to view all of the data.

```
10  OUTPUT 707;":WAVEFORM:VIEW ALL"
20  END
```

**Table 25-2** **Waveform View Parameters**

| Source/Parameter | ALL | MAIN | WINDow |
|---|---|---|---|
| **CHANNEL** | All data | Main time base | Delayed time base |
| **MEMORY** | All data | Memory time base | Memory time base |
| **FUNCTION** | All data | All data | All data |

Query

```
:WAVeform:VIEW?
```

The :WAVeform:VIEW? query returns the currently selected view.

Returned Format

```
[:WAVeform:VIEW] {ALL | MAIN | WINDow}<NL>
```

**Example**

This example returns the current view setting to the string variable, Setting$, then prints the contents of the variable to the computer's screen.

```
10  DIM Setting$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:VIEW?"
30  ENTER 707;Setting$
40  PRINT Setting$
50  END
```

# XDISplay?

Query

```
:WAVeform:XDISplay?
```

The :WAVeform:XDISplay? query returns the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. For VERSus type waveforms, it is the value at the center of the X-axis of the display. This value is treated as a double precision 64-bit floating point number.

Returned Format

```
[:WAVeform:XDISplay] <value><NL>
```

<value> A real number representing the X-axis value at the left edge of the display.

**Example**

This example returns the X-axis value at the left edge of the display to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:XDISPLAY?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

## XINCrement?

**Query**

```
:WAVeform:XINCrement?
```

The :WAVeform:XINCrement? query returns the duration between consecutive data points for the currently specified waveform source. For time domain waveforms, this is the time difference between consecutive data points. For VERSus type waveforms, this is the duration between levels on the X axis. For voltage waveforms, this is the voltage corresponding to one level.

**Returned Format**

```
[:WAVeform:XINCrement] <value><NL>
```

<value> A real number representing the duration between data points on the X axis.

**Example**

This example places the current X-increment value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:XINCREMENT?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

**See Also**

You can obtain the X-increment value through the :WAVeform:PREamble? query.

# XORigin?

Query

```
:WAVeform:XORigin?
```

The :WAVeform:XORigin? query returns the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. For VERSus type waveforms, it is the X-axis value at level zero. For voltage waveforms, it is the voltage at level zero. The value returned by this query is treated as a double precision 64-bit floating point number.

Returned Format

```
[:WAVeform:XORigin] <value><NL>
```

<value> A real number representing the X-axis value of the first data point in the data record.

**Example**

This example places the current X-origin value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:XORIGIN?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

**See Also**

You can obtain the X-origin value through the :WAVeform:PREamble? query.

# XRANge?

**Query**

```
:WAVeform:XRANge?
```

The :WAVeform:XRANge? query returns the X-axis duration of the displayed waveform. For time domain waveforms, it is the duration of the time across the display. For VERSus type waveforms, it is the duration of the waveform that is displayed on the X axis.

**Returned Format**

```
[:WAVeform:XRANge] <value><NL>
```

<value> A real number representing the X-axis duration of the displayed waveform.

**Example**

This example returns the X-axis duration of the displayed waveform to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:XRANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

## XREFerence?

Query

```
:WAVeform:XREFerence?
```

The :WAVeform:XREFerence? query returns the data point or level associated with the X-origin data value. It is at this data point or level that the X origin is defined. In this oscilloscope, the value is always zero.

Returned Format

```
[:WAVeform:XREFerence] 0<NL>
```

**Example**

This example places the current X-reference value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:XREFERENCE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

**See Also**

You can obtain the X-reference value through the :WAVeform:PREamble? query.

# XUNits?

Query

```
:WAVeform:XUNits?
```

The :WAVeform:XUNits? query returns the X-axis units of the currently selected waveform source. The currently selected source may be a channel, function, or waveform memory.

Returned Format

```
[:WAVeform:XUNits] {UNKNown | VOLT | SECond | CONStant | AMP
| DECibels | HERTz | WATT}<NL>
```

**Example**

This example returns the X-axis units of the currently selected waveform source to the string variable, Unit$, then prints the contents of the variable to the computer's screen.

```
10  DIM Unit$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:XUNITS?"
30  ENTER 707;Unit$
40  PRINT Unit$
50  END
```

# YDISplay?

Query

```
:WAVeform:YDISplay?
```

The :WAVeform:YDISplay? query returns the Y-axis value at the center of the display. For voltage waveforms, it is the voltage at the center of the display.

Returned Format

```
[:WAVeform:YDISplay] <value><NL>
```

<value> A real number representing the Y-axis value at the center of the display.

**Example**

This example returns the current Y-display value to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;"":WAVEFORM:YDISPLAY?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

## YINCrement?

Query

```
:WAVeform:YINCrement?
```

The :WAVeform:YINCrement? query returns the y-increment voltage value for the currently specified source. This voltage value is the voltage difference between two adjacent waveform data digital codes. Adjacent digital codes are codes that differ by one least significant bit. For example, the digital codes 24680 and 24681 vary by one least significant bit.

- For BYTE and WORD data, and voltage waveforms, it is the voltage corresponding to one least significant bit change.
- For ASCii data format, the YINCrement is the full scale voltage range covered by the A/D converter.

Returned Format

```
[:WAVeform:YINCrement] <real_value><NL>
```

`<real_value>` A real number in exponential format.

**Example**

This example places the current Y-increment value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:YINCREMENT?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

**See Also**

For more information on BYTE and WORD formats see "Understanding WORD and BYTE Formats" on page 25-28.

You can also obtain the Y-increment value through the :WAVeform:PREamble? query.

# YORigin?

Query

```
:WAVeform:YORigin?
```

The :WAVeform:YORigin? query returns the y-origin voltage value for the currently specified source. The voltage value returned is the voltage value represented by the waveform data digital code 00000.

- For BYTE and WORD data, and voltage waveforms, it is the voltage at digital code zero.
- For ASCii data format, the YORigin is the Y-axis value at the center of the data range. Data range is returned in the Y increment.

Returned Format

```
[:WAVeform:YORigin] <real_value><NL>
```

`<real_value>` A real number in exponential format.

**Example**

This example places the current Y-origin value in the numeric variable, Center, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:YORIGIN?"
30  ENTER 707;Center
40  PRINT Center
50  END
```

**See Also**

For more information on BYTE and WORD formats see "Understanding WORD and BYTE Formats" on page 25-28.

You can obtain the Y-origin value through the :WAVeform:PREamble? query.

# YRANge?

**Query**

```
:WAVeform:YRANge?
```

The :WAVeform:YRANge? query returns the Y-axis duration of the displayed waveform. For voltage waveforms, it is the voltage across the entire display.

**Returned Format**

```
[:WAVeform:YRANge] <value><NL>
```

`<value>` A real number representing the Y-axis duration of the displayed waveform.

**Example**

This example returns the current Y-range value to the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:YRANGE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

## YREFerence?

**Query**

```
:WAVeform:YREFerence?
```

The :WAVeform:YREFerence? query returns the y-reference voltage value for the currently specified source. It is at this level that the Y origin is defined. In this oscilloscope, the value is always zero.

**Returned Format**

```
[:WAVeform:YREFerence]<integer_value><NL>
```

`<integer_value>` Always 0.

**Example**

This example places the current Y-reference value for the currently specified source in the numeric variable, Value, then prints the contents of the variable to the computer's screen.

```
10  OUTPUT 707;":SYSTEM:HEADER OFF"!Response headers off
20  OUTPUT 707;":WAVEFORM:YREFERENCE?"
30  ENTER 707;Value
40  PRINT Value
50  END
```

**See Also**

For more information on BYTE and WORD formats see "Understanding WORD and BYTE Formats" on page 25-28.

You can obtain the Y-reference value through the :WAVeform:PREamble? query.

# YUNits?

**Query**

```
:WAVeform:YUNits?
```

The :WAVeform:YUNits? query returns the Y-axis units of the currently selected waveform source. The currently selected source may be a channel, function, or waveform memory.

**Returned Format**

```
[:WAVeform:YUNits] {UNKNown | VOLT | SECond | CONStant | AMP
| DECibels}<NL>
```

**Example**

This example returns the Y-axis units of the currently selected waveform source to the string variable, Unit$, then prints the contents of the variable to the computer's screen.

```
10  DIM Unit$[50]!Dimension variable
20  OUTPUT 707;":WAVEFORM:YUNITS?"
30  ENTER 707;Unit$
40  PRINT Unit$
50  END
```

# Waveform Memory Commands

# Waveform Memory Commands

The Waveform Memory Subsystem commands let you save and display waveforms, memories, and functions. These Waveform Memory commands and queries are implemented in the Infiniium Oscilloscopes:

- DISPlay
- LFFile
- LOAD
- SAVE
- XOFFset
- XRANge
- YOFFset
- YRANge

**<N> in WMEMory<N> Indicates the Waveform Memory Number**

In Waveform Memory commands, the <N> in WMEMory<N> represents the waveform memory number (1-4).

# DISPlay

**Command**

```
:WMEMory<N>:DISPlay {{ON|1} | {OFF|0}}
```

The :WMEMory<N>:DISPlay command enables or disables the viewing of the selected waveform memory.

<N> The memory number is an integer from 1 to 4.

**Example**

This example turns on the waveform memory 1 display.

```
10  OUTPUT 707;":WMEMORY1:DISPLAY ON"
20  END
```

**Query**

```
:WMEMory<N>:DISPlay?
```

The :WMEMory<N>:DISPlay? query returns the state of the selected waveform memory.

**Returned Format**

[:WMEMory<N>:DISPlay] {1 | 0}<NL>

# LOAD

Command

```
:WMEMory<N>:LOAD <file_name>

:WMEMory<N>:LFFile <file_name>

:WMEMory<N>:LOADFROMFILE <file_name>
```

The :WMEMory<N>:LOAD command loads an oscilloscope waveform memory location with a waveform from a file that has an internal waveform format (extension .wfm) or a verbose/yvalues waveform format (extension .txt). You can load the file from either the c: or a: drive. See the examples below.

The oscilloscope assumes that the default path for waveforms is c:\scope\data. To use a different path, specify the path and file name completely.

`<N>` The memory number is an integer from 1 to 4.

`<file_name>` A quoted string which specifies the file to load, and has either a .wfm or .txt extension.

**Examples**

This example loads waveform memory 4 with a file that has the internal waveform format.

```
10  OUTPUT 707;":WMEMORY4:LOAD ""c:\scope\data\waveform.wfm"""
20  END
```

This example loads waveform memory 3 with a file that has the internal waveform format and is stored on the floppy drive.

```
10  OUTPUT 707;":WMEMORY3:LOAD ""a:\waveform.wfm"""
20  END
```

**Related Commands**

:DISK:LOAD, :DISK:STORe
:WMEMory<N>:LoadFromFile <file_name>

# SAVE

Command

```
:WMEMory<N>:SAVE {CHANnel<N> | WMEMory<N> |
FUNCtion<N>}
```

The :WMEMory<N>:SAVE command stores the specified channel, waveform memory, or function to the waveform memory. The channel or function must be displayed (DISPlay must be set to ON) or an error status is returned. You can save waveforms to waveform memories regardless of whether the waveform memory is displayed or not.

<N> CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

Integers, 1 - 4, representing the selected function or waveform memory.

**Example**

This example saves channel 1 to waveform memory 4.

```
10  OUTPUT 707;":WMEMORY4:SAVE chan1"
20  END
```

# XOFFset

Command

```
:WMEMory<N>:XOFFset <offset_value>
```

The :WMEMory<N>:XOFFset command sets the x-axis, horizontal position for the selected waveform memory's display scale. The position is referenced to center screen.

`<N>` The memory number is an integer from 1 to 4.

`<offset_value>` A real number for the horizontal offset (position) value.

**Example**

This example sets the X-axis, horizontal position for waveform memory 3 to 0.1 seconds (100 ms).

```
10  OUTPUT 707;":WMEMORY3:XOFFSET 0.1"
20  END
```

Query

```
:WMEMory<N>:XOFFset?
```

The :WMEMory<N>:XOFFset? query returns the current X-axis, horizontal position for the selected waveform memory.

Returned Format

```
[:WMEMory<N>:XOFFset] <offset_value><NL>
```

# XRANge

**Command**

```
:WMEMory<N>:XRANge <range_value>
```

The :WMEMory<N>:XRANge command sets the X-axis, horizontal range for the selected waveform memory's display scale. The horizontal scale is the horizontal range divided by 10.

`<N>` The memory number is an integer from 1 to 4.

`<range_value>` A real number for the horizontal range value.

**Example**

This example sets the X-axis, horizontal range of waveform memory 2 to 435 microseconds.

```
10  OUTPUT 707;":WMEMORY2:XRANGE 435E-6"
20  END
```

**Query**

```
:WMEMory<N>:XRANge?
```

The :WMEMory<N>:XRANge? query returns the current X-axis, horizontal range for the selected waveform memory.

**Returned Format**

```
[:WMEMory<N>:XRANge] <range_value><NL>
```

# YOFFset

Command

```
:WMEMory<N>:YOFFset <offset_value>
```

The :WMEMory<N>:YOFFset command sets the Y-axis (vertical axis) offset for the selected waveform memory.

`<N>` The memory number is an integer from 1 to 4.

`<offset_value>` A real number for the vertical offset value.

**Example**

This example sets the Y-axis (vertical) offset of waveform memory 2 to 0.2V.

```
10  OUTPUT 707;":WMEMORY2:YOFFSET 0.2"
20  END
```

Query

```
:WMEMory<N>:YOFFset?
```

The :WMEMory<N>:YOFFset? query returns the current Y-axis (vertical) offset for the selected waveform memory.

Returned Format

```
[:WMEMory<N>:YOFFset] <offset_value><NL>
```

# YRANge

**Command**

```
:WMEMory<N>:YRANge <range_value>
```

The :WMEMory<N>:YRANge command sets the Y-axis, vertical range for the selected memory. The vertical scale is the vertical range divided by 8.

`<N>` The memory number is an integer from 1 to 4.

`<range_value>` A real number for the vertical range value.

**Example**

This example sets the Y-axis (vertical) range of waveform memory 3 to 0.2 volts.

```
10  OUTPUT 707;":WMEMORY3:YRANGE 0.2"
20  END
```

**Query**

```
:WMEMory<N>:YRANge?
```

The :WMEMory<N>:YRANge? query returns the Y-axis, vertical range for the selected memory.

**Returned Format**

```
[:WMEMory<N>:YRANge]<range_value><NL>
```

# Infiniium and HP 547XX Digitizing Oscilloscopes Language Compatibility

# Infiniium and HP 547XX Digitizing Oscilloscopes Language Compatibility

When developing new programs, you should use the Infiniium (HP 548XX) command language, as documented in the chapters in this manual. When using existing programs (that you do not want to modify) with Infiniium Oscilloscopes, the HP 547XX and HP 545XX command sets are provided as built-in languages. See "Some HP 547XX Commands are New" on the next page.

- This chapter describes language compatibility with HP 547XX oscilloscopes.
- Chapter 26 describes language compatibility with HP 545XX oscilloscopes.

The built-in command languages make your use of the Infiniium Oscilloscopes compatible with previously designed oscilloscopes — ones that you may already be used to operating. Being able to choose another command language is beneficial if you want to use existing programs on Infiniium Oscilloscopes without having to modify your programs.

The built-in HP 547XX and HP 545XX languages can also be helpful if you are familiar with one or both of them, and want to continue using that language on Infiniium Oscilloscopes.

### Selecting a Command Language

Use the :SYSTem:LANGuage command to select either the HP 548XX, HP 547XX, or HP 545XX command language built into the Infiniium Oscilloscopes. The HP 548XX command language is the default.

### Command Language Tables

There are some differences between the built-in command languages. The tables in this chapter show these differences (if any exist), and the relationships between the command languages for HP 548XX Infiniium Oscilloscopes and HP 547XX Digitizing Oscilloscopes. If a command is supported only on the 54846A, 54845A, and 54835A, it is noted.

### Some HP 547XX Commands are New

Some HP 547XX commands are new. That is, they were not in the original command language, but they have been added to the HP 547XX language support for HP 548XX Oscilloscopes.

### What the * Symbol Means

In the command tables, the " * " symbol after a command indicates the command is newly supported.

**See Also**

See Chapter 26 for language compatibility between Infiniium and HP 545XX Oscilloscopes.

## To select a command language

In Infiniium Oscilloscopes, you can select one of the supported command languages either over the GPIB or from the oscilloscope front panel.

- To select one of the command languages over the GPIB, enter the appropriate command for the oscilloscope type you are using:

  :SYSTem:LANGuage HP547XX

  :SYSTem:LANGuage HP548XX

  (This chapter describes the HP 547XX and HP 548XX command language compatibility.)

  :SYSTem:LANGuage HP545XX

  (Use the information in Chapter 26 for the HP 545XX Oscilloscopes command language compatibility.)

- To select one of the command languages from the oscilloscope front panel, select Utilities, Remote Interface, and Select Language. Then choose HP545XX, HP547XX, or HP548XX.

# Acquisition System Command Language Compatibility

### HP 548XX and HP 547XX ACQuire Commands

| HP 548XX | HP 547XX |
|---|---|
| :ACQuire:AVERage | Note |
| :ACQuire:AVERage:COUNt | Note |
| :ACQuire:BWLimit | :ACQuire:BWLimit |
| :ACQuire:COMPlete | :ACQuire:COMPlete |
| :ACQuire:COMPlete:STATe | :ACQuire:COMPlete:STATe |
| :ACQuire:CONFig (54846/45A/35A only) | :ACQuire:CONFig * |
| :ACQuire:COUNt | :ACQuire:COUNt |
| :ACQuire:INTerpolate | :ACQuire:INTerpolate |
| :ACQuire:MODE | :ACQuire:MODE |
| :ACQuire:POINts | :ACQuire:POINts |
| :ACQuire:SRATe | :ACQuire:SRATe |
| Note | :ACQuire:TYPE |

Note: Command not supported on this oscilloscope.

# Calibration Command Language Compatibility

### HP 548XX and HP 547XX CALibrate Commands

| HP 548XX | HP 547XX |
|---|---|
| Note | :CALibrate:BEST:CANCel |
| Note | :CALibrate:BEST:CONTinue |
| Note | :CALibrate:BEST:DATA |
| Note | :CALibrate:BEST:STARt |
| Note | :CALibrate:BEST:STATus? |
| :CALibrate:CANCel | Note |
| :CALibrate:CONTinue | Note |
| Note | :CALibrate:FRAMe:CANCel |
| Note | :CALibrate:FRAMe:CONTinue |
| Note | :CALibrate:FRAMe:DATA |
| Note | :CALibrate:FRAMe:DONE? |
| Note | :CALibrate:FRAMe:LABel |
| Note | :CALibrate:FRAMe:STARt |
| Note | :CALibrate:FRAMe:MEMory? |
| :CALibrate:MPRotect | Note |
| :CALibrate:OUTPut | :CALibrate:OUTPut |
| Note | :CALibrate:PLUGin:TIME? |
| Note | :CALibrate:PLUGin:CANCel |
| Note | :CALibrate:PLUGin:CONTinue |
| Note | :CALibrate:PLUGin:DONE? |
| Note | :CALibrate:PLUGin:MEMory? |
| Note | :CALibrate:PLUGin:STARt |
| Note | :CALibrate:PLUGin:TIME? |
| :CALibrate:SDONe? | Note |
| :CALibrate:SKEW | :CALibrate:SKEW |
| :CALibrate:STARt | Note |
| :CALibrate:STATus? | :CALibrate:STATus? |

Note: Command not supported on this oscilloscope.

# Channel Command Language Compatibility

### HP 548XX and HP 547XX CHANnel Commands

| HP 548XX | HP 547XX |
|---|---|
| :CHANnel<N>:BWLimit | :CHANnel<N>:BWLimit |
| :CHANnel<N>:DISPlay | :CHANnel<N>:DISPlay |
| :CHANnel<N>:ECL | :CHANnel<N>:ECL * |
| :CHANnel<N>:INPut | :CHANnel<N>:INPut |
| :CHANnel<N>:OFFSet | :CHANnel<N>:OFFSet |
| :CHANnel<N>:PROBe | :CHANnel<N>:PROBe |
| Note | :PROBe:INPut |
| :CHANnel<N>:PROBe:EGAin | :CHANnel<N>:PROBe:EGAin * |
| :CHANnel<N>:PROBe:EOFFset | :CHANnel<N>:PROBe:EOFFset * |
| :CHANnel<N>:PROBe:SKEW | :CHANnel<N>:PROBe:SKEW * |
| :CHANnel<N>:PROTection? (54846/45A/35A only) | :CHANnel<N>:PROTection? * (54846A/45A/35A only) |
| :CHANnel<N>:PROTection:CLEar (54846A/45A/35A only) | :CHANnel<N>:PROTection:CLEar * (54846A/45A/35A only) |
| :CHANnel<N>:RANGe | :CHANnel<N>:RANGe |
| Note | :CHANnel<N>:SCALe |
| :CHANnel<N>:TTL | :CHANnel<N>:TTL * |
| :CHANnel<N>:UNITs | :CHANnel<N>:UNITs |

Note: Command not supported on this oscilloscope.

# Disk Command Language Compatibility

### HP 548XX and HP 547XX DISK Commands

| HP 548XX | HP 547XX |
|---|---|
| :DISK:CDIRectory | :DISK:CDIRectory * |
| :DISK:DELete | :DISK:DELete |
| :DISK:DIRectory? | :DISK:DIRectory? |
| :DISK:GetFILe? | :DISK:GetFILe? * |
| :DISK:LOAD | :DISK:LOAD |
| :DISK:MDIRectory | :DISK:MDIRectory * |
| :DISK:PWD? | :DISK:PWD? * |
| :DISK:STORe | :DISK:STORe |

# Display Command Language Compatibility

### HP 548XX and HP 547XX DISPlay Commands

| HP 548XX | HP 547XX |
|---|---|
| :DISPlay:COLumn | :DISPlay:COLumn |
| :DISPlay:CONNect | :DISPlay:CONNect * |
| :DISPlay:DATA? | :DISPlay:DATA |
| :DISPlay:DCOLor | :DISPlay:DCOLor |
| Note | :DISPlay:DWAVeform |
| :DISPlay:GRATicule | :DISPlay:GRATicule |
| :DISPlay:GRATicule:INTensity | :DISPlay:GRATicule:INTensity * |
| :DISPlay:LAYout:MBAR | :DISPlay:LAYout:MBAR * |
| :DISPlay:LAYout:MRESults | :DISPlay:LAYout:MRESults * |
| :DISPlay:LINE | :DISPlay:LINE |
| :DISPlay:PERSistence | :DISPlay:PERSistence |
| :DISPlay:PERSistence:TIME | :DISPlay:PERSistence:TIME |
| :DISPlay:ROW | :DISPlay:ROW |
| :DISPlay:SCOLor | :DISPlay:SCOLor |
| :DISPlay:SSAVer | :DISPlay:SSAVer * |
| :DISPlay:SSAVer:AAFTer | :DISPlay:SSAVer:AAFTer * |
| :DISPlay:STRing | :DISPlay:STRing |
| :DISPlay:TEXT | :DISPlay:TEXT |

Note: Command not supported on this oscilloscope.

# External Command Language Compatibility

### HP 548XX and HP 547XX EXTernal Commands

| HP 548XX | HP 547XX |
|---|---|
| :EXTernal:BWLimit | :EXTernal:BWLimit * |
| :EXTernal:INPut | :EXTernal:INPut * |
| :EXTernal:PROBe | :EXTernal:PROBe * |
| :EXTernal:PROBe:EGAin | :EXTernal:PROBe:EGAin * |
| :EXTernal:PROBe:EOFFset | :EXTernal:PROBe:EOFFset * |
| :EXTernal:PROBe:SKEW | :EXTernal:PROBe:SKEW * |
| :EXTernal:RANGe | :EXTernal:RANGe * |
| :EXTernal:UNITs | :EXTernal:UNITs * |

# FFT Command Language Compatibility

Note: For HP 547XX Oscilloscopes, FFT is now FUNCtion3.
This subsystem is not implemented for HP 548XX Oscillscopes.

**HP 547XX FFT Commands**

| HP 548XX | HP 547XX |
|---|---|
| Note | Note: FFT is now FUNCtion3 |
| Note | :FFT:DISPlay |
| Note | :FFT:FREQuency |
| Note | :FFT:OFFSet |
| Note | :FFT:RANGe |
| Note | :FFT:SOURce |
| Note | :FFT:SPAN |
| Note | :FFT:WINDow |

Note: Command not supported on this oscilloscope.

# Function Command Language Compatibility

### HP 548XX and HP 547XX FUNCtion Commands

| HP 548XX | HP 547XX |
|---|---|
| :FUNCtion<N>? | :FUNCtion<N>? * |
| :FUNCtion<N>:ADD | :FUNCtion<N>:ADD |
| :FUNCtion<N>:DIFF | :FUNCtion<N>:DIFF |
| :FUNCtion<N>:DISPlay | :FUNCtion<N>:DISPlay |
| :FUNCtion<N>:DIVide | :FUNCtion<N>:DIVide |
| :FUNCtion<N>:FFT:FREQuency | :FUNCtion<N>:FFT:FREQuency |
| :FUNCtion<N>:FFT:RESolution? | :FUNCtion<N>:FFT:RESolution |
| Note | :FUNCtion<N>:FFT:SPAN |
| :FUNCtion<N>:FFT:WINDow | :FUNCtion<N>:FFT:WINDow |
| :FUNCtion<N>:FFTMagnitude | :FUNCtion<N>:FFTMagnitude |
| :FUNCtion<N>:HORizontal | :FUNCtion<N>:HORizontal |
| :FUNCtion<N>:HORizontal:POSition | :FUNCtion<N>:HORizontal:POSition |
| :FUNCtion<N>:HORizontal:RANGe | :FUNCtion<N>:HORizontal:RANGe |
| :FUNCtion<N>:INTegrate | :FUNCtion<N>:INTegrate |
| :FUNCtion<N>:INVert | :FUNCtion<N>:INVert |
| :FUNCtion<N>:MAGNify | :FUNCtion<N>:MAGNify |
| :FUNCtion<N>:MAXimum | :FUNCtion<N>:MAXimum |
| :FUNCtion<N>:MINimum | :FUNCtion<N>:MINimum |
| :FUNCtion<N>:MULTiply | :FUNCtion<N>:MULTiply |
| :FUNCtion<N>:OFFSet | :FUNCtion<N>:OFFSet |
| Note | :FUNCtion<N>:ONLY |
| :FUNCtion<N>:RANGe | :FUNCtion<N>:RANGe |
| :FUNCtion<N>:SUBTract | :FUNCtion<N>:SUBTract |
| :FUNCtion<N>:VERSus | :FUNCtion<N>:VERSus |
| :FUNCtion<N>:VERTical | :FUNCtion<N>:VERTical |
| :FUNCtion<N>:VERTical:OFFSet | :FUNCtion<N>:VERTical:OFFSet |
| :FUNCtion<N>:VERTical:RANGe | :FUNCtion<N>:VERTical:RANGe |

Note: Command not supported on this oscilloscope.

# Hardcopy Command Language Compatibility

**HP 548XX and HP 547XX HARDcopy Commands**

| HP 548XX | HP 547XX |
|---|---|
| :HARDcopy:AREA | :HARDcopy:AREA |
| :HARDcopy:DPRinter | :HARDcopy:DPRinter * |
| :HARDcopy:FACTors | :HARDcopy:FACTors |
| :HARDcopy:IMAGe | :HARDcopy:IMAGe * |
| :HARDcopy:PRINters? | :HARDcopy:PRINters? * |

# Limit Test Command Language Compatibility

Limit TESt commands do not apply to HP 548XX Oscilloscopes.

# Marker Command Language Compatibility

### HP 548XX and HP 547XX MARKer Commands

| HP 548XX | HP 547XX |
|---|---|
| :MARKer:CURSor? | :MARKer:CURSor? |
| :MARKer:MEASurement:READout | :MARKer:MEASurement:READout |
| :MARKer:MODE | :MARKer:MODE |
| :MARKer:TDELta? | :MARKer:TDELta? |
| :MARKer:TSTArt | :MARKer:TSTArt |
| :MARKer:TSTOp | :MARKer:TSTOp |
| :MARKer:VDELta? | :MARKer:VDELta? |
| :MARKer:VSTArt | :MARKer:VSTArt |
| :MARKer:VSTOp | :MARKer:VSTOp |
| :MARKer:X1Position | :MARKer:X1Position |
| :MARKer:X1Y1source | :MARKer:X1Y1source |
| :MARKer:X2Position | :MARKer:X2Position |
| :MARKer:X2Y2source | :MARKer:X2Y2source |
| :MARKer:XDELta? | :MARKer:XDELta? |
| :MARKer:Y1Position | :MARKer:Y1Position |
| :MARKer:Y2Position | :MARKer:Y2Position |
| :MARKer:YDELta? | :MARKer:YDELta? |

### Sources for MARKer Commands

Sources for the MARKer commands can be CHANnel<N>, FUNCtion<N>, or WMEMory<N>.

<N> CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

Integers, 1 - 4, representing the selected function or waveform memory.

# Measure Command Language Compatibility

### HP 548XX and HP 547XX MEASure Commands

| HP 548XX | HP 547XX |
|---|---|
| :MEASure:ALL? | :MEASure:ALL? * |
| :MEASure:CLEar | :MEASure:CLEar * |
| :MEASure:DEFine | :MEASure:DEFine |
| :MEASure:DELTatime | :MEASure:DELTatime |
| :MEASure:DUTYcyclelDUT | :MEASure:DUTYcyclelDUT |
| :MEASure:FALLtime | :MEASure:FALLtime |
| :MEASure:FFT:DFRequency | :MEASure:FFT:DFRequency |
| :MEASure:FFT:DMAGnitude | :MEASure:FFT:DMAGnitude |
| :MEASure:FFT:FREQuency | :MEASure:FFT:FREQuency |
| :MEASure:FFT:MAGNitude | :MEASure:FFT:MAGNitude |
| :MEASure:FFT:PEAK1 | :MEASure:FFT:PEAK1 |
| :MEASure:FFT:PEAK2 | :MEASure:FFT:PEAK2 |
| :MEASure:FFT:THReshold | :MEASure:FFT:THReshold |
| :MEASure:FREQuency | :MEASure:FREQuency |
| :MEASure:NWIDth | :MEASure:NWIDth |
| :MEASure:OVERshoot | :MEASure:OVERshoot |
| :MEASure:PERiod | :MEASure:PERiod |
| :MEASure:PREShoot | :MEASure:PREShoot |
| :MEASure:PWIDth | :MEASure:PWIDth |
| :MEASure:RESults? | :MEASure:RESults? |
| :MEASure:RISetime | :MEASure:RISetime |
| :MEASure:SCRatch | :MEASure:SCRatch |
| :MEASure:SENDvalid | :MEASure:SENDvalid |
| :MEASure:SOURce | :MEASure:SOURce |
| :MEASure:STATistics | :MEASure:STATistics |
| :MEASure:TEDGe | :MEASure:TEDGe |
| :MEASure:TMAX | :MEASure:TMAX |
| :MEASure:TMIN | :MEASure:TMIN |
| :MEASure:TVOLt | :MEASure:TVOLt |
| :MEASure:VAMPlitude | :MEASure:VAMPlitude |
| :MEASure:VAVerage | :MEASure:VAVerage |
| :MEASure:VBASe | :MEASure:VBASe |
| :MEASure:VLOWer | :MEASure:VLOWer |
| :MEASure:VMAX | :MEASure:VMAX |
| :MEASure:VMIDdle | :MEASure:VMIDdle |
| :MEASure:VMIN | :MEASure:VMIN |
| :MEASure:VPP | :MEASure:VPP |
| :MEASure:VRMS | :MEASure:VRMS |
| :MEASure:VTIMe | :MEASure:VTIMe |
| :MEASure:VTOP | :MEASure:VTOP |
| :MEASure:VUPPer | :MEASure:VUPPer\|VUP |

# Multiple Memory Command Language Compatibility

Multiple MEMory commands do not apply to HP 548XX or HP 547XX Oscilloscopes.

# Memory Test Command Language Compatibility

Memory TESt commands do not apply to HP 548XX Oscilloscopes.

# Pixel Memory Command Language Compatibility

Pixel MEMory commands do not apply to HP 548XX or HP 547XX Oscilloscopes.

# Self-Test Command Language Compatibility

**HP 548XX and HP 547XX SELFtest Commands**

| HP 548XX | HP 547XX |
|---|---|
| :SELFtest:ASET | :SELFtest:ASET * |
| :SELFtest:SCOPETEST | :SELFtest:SCOPETEST * |

# Sequential Command Language Compatibility

SEQuential commands do not apply to HP 548XX or HP 547XX Oscilloscopes.

# System Command Language Compatibility

### HP 548XX and HP 547XX SYSTem Commands

| HP 548XX | HP 547XX |
|---|---|
| :SYSTem:DATE | :SYSTem:DATE |
| :SYSTem:DEBug | :SYSTem:DEBug * |
| :SYSTem:DSP | :SYSTem:DSP |
| :SYSTem:ERRor? | :SYSTem:ERRor? |
| :SYSTem:HEADer | :SYSTem:HEADer |
| :SYSTem:HELP:HEADers? | :SYSTem:HELP:HEADers? * |
| :SYSTem:HPIB:ADDRess? | :SYSTem:HPIB:ADDRess? * |
| :SYSTem:LANGuage | :SYSTem:LANGuage |
| :SYSTem:LONGform | :SYSTem:LONGform |
| :SYSTem:SETup | :SYSTem:SETup |
| :SYSTem:TIME | :SYSTem:TIME |

# Time Base Command Language Compatibility

### HP 548XX and HP 547XX TIMebase Commands

| HP 548XX | HP 547XX |
|---|---|
| :TIMebase:DELay | :TIMebase:DELay |
| :TIMebase:POSition | :TIMebase:POSition |
| :TIMebase:RANGe | :TIMebase:RANGe |
| :TIMebase:REFerence | :TIMebase:REFerence |
| :TIMebase:SCALe | :TIMebase:SCALe |
| :TIMebase:SETup? | :TIMebase:SETup? * |
| :TIMebase:VIEW | :TIMebase:VIEW |
| :TIMebase:WINDow:DELay | :TIMebase:WINDow:DELay |
| :TIMebase:WINDow:POSition | :TIMebase:WINDow:POSition |
| :TIMebase:WINDow:RANGe | :TIMebase:WINDow:RANGe |
| :TIMebase:WINDow:SCALe | :TIMebase:WINDow:SCALe |
| :TIMebase:WINDow:SOURce | :TIMebase:WINDow:SOURce |

# Trigger Command Language Compatibility

### HP 548XX and HP 547XX TRIGger Commands

| HP 548XX | HP 547XX |
|---|---|
| :TRIGger:ADVanced | Note |
| :DELay:EDLY:ARM:SLOPe | Note |
| :DELay:EDLY:ARM:SOURce | Note |
| :DELay:EDLY:EVENt:DELay | Note |
| :DELay:EDLY:EVENt:SLOPe | Note |
| :DELay:EDLY:EVENt:SOURce | Note |
| :DELay:EDLY:TRIGger:SLOPe | Note |
| :DELay:EDLY:TRIGger:SOURce | Note |
| :DELay:MODE | Note |
| :DELay:TDLY:ARM:SLOPe | Note |
| :DELay:TDLY:ARM:SOURce | Note |
| :DELay:TDLY:DELay | Note |
| :DELay:TDLY:TRIGger:SLOPe | Note |
| :DELay:TDLY:TRIGger:SOURce | Note |
| :MODE | Note |
| :PATTern:CONDition | Note |
| :PATTern:LOGic | Note |
| :STATe:CLOCk | Note |
| :STATe:CONDition | Note |
| :STATe:LOGic | Note |
| :STATe:LTYPe | Note |
| :STATe:SLOPe | Note |
| :TV:MODE | Note |
| :TV:STV:FIELd | Note |
| :TV:STV:LINE | Note |
| :TV:STV:SOURce | Note |
| :TV:STV:SPOLarity | Note |
| :TV:STV:STANdard | Note |
| :TV:UDTV:EDGE | Note |
| :TV:UDTV:ENUMber | Note |
| :TV:UDTV:PGTHan | Note |
| :TV:UDTV:PLTHan | Note |
| :TV:UDTV:POLarity | Note |
| :TV:UDTV:SOURce | Note |
| :VIOLation:MODE | :VIOLation:MODE * |
| :VIOLation:PWIDth:DIRection | :VIOLation:PWIDth:DIRection * |
| :VIOLation:PWIDth:POLarity | :VIOLation:PWIDth:POLarity * |
| :VIOLation:PWIDth:SOURce | :VIOLation:PWIDth:SOURce * |
| :VIOLation:PWIDth:WIDTh | :VIOLation:PWIDth:WIDTh * |
| :VIOLation:RUNT:DIRection | :VIOLation:RUNT:DIRection * |
| :VIOLation:RUNT:SOURce:HTHReshold | :VIOLation:RUNT:SOURce:HTHReshold * |
| :VIOLation:RUNT:SOURce:LTHReshold | :VIOLation:RUNT:SOURce:LTHReshold * |
| :VIOLation:RUNT:TQUalified | :VIOLation:RUNT:TQUalified * |
| :VIOLation:SETup:HOLD:CSOurce:EDGE | :VIOLation:SETup:HOLD:CSOurce:EDGE * |

| | |
|---|---|
| :VIOLation:SETup:HOLD:CSOurce:LEVel | :VIOLation:SETup:HOLD:CSOurce:LEVel * |
| :VIOLation:SETup:HOLD:DSOurce:HTHReshold | :VIOLation:SETup:HOLD:DSOurce:HTHReshold * |
| :VIOLation:SETup:HOLD:DSOurce:LTHReshold | :VIOLation:SETup:HOLD:DSOurce:LTHReshold * |
| :VIOLation:SETup:HOLD:TIME | :VIOLation:SETup:HOLD:TIME * |
| :VIOLation:SETup:MODE | :VIOLation:SETup:MODE * |
| :VIOLation:SETup:SETup:CSOurce:EDGE | :VIOLation:SETup:SETup:CSOurce:EDGE * |
| :VIOLation:SETup:SETup:CSOurce:LEVel | :VIOLation:SETup:SETup:CSOurce:LEVel * |
| :VIOLation:SETup:SETup:DSOurce:HTHReshold | :VIOLation:SETup:SETup:DSOurce:HTHReshold * |
| :VIOLation:SETup:SETup:DSOurce:LTHReshold | :VIOLation:SETup:SETup:DSOurce:LTHReshold * |
| :VIOLation:SETup:SETup:TIME | :VIOLation:SETup:SETup:TIME * |
| :VIOLation:SETup:SHOLd:CSOurce:EDGE | :VIOLation:SETup:SHOLd:CSOurce:EDGE * |
| :VIOLation:SETup:SHOLd:CSOurce:LEVel | :VIOLation:SETup:SHOLd:CSOurce:LEVel * |
| :VIOLation:SETup:SHOLd:DSOurce:HTHReshold | :VIOLation:SETup:SHOLd:DSOurce:HTHReshold * |
| :VIOLation:SETup:SHOLd:DSOurce:LTHReshold | :VIOLation:SETup:SHOLd:DSOurce:LTHReshold * |
| :VIOLation:SETup:SHOLd:HOLDTIME | :VIOLation:SETup:SHOLd:HOLDTIME * |
| :VIOLation:SETup:SHOLd:HTIMe | :VIOLation:SETup:SHOLd:HTIMe * |
| :VIOLation:SETup:SHOLd:SETUPTIME | :VIOLation:SETup:SHOLd:SETUPTIME * |
| :VIOLation:SETup:SHOLd:STIMe | :VIOLation:SETup:SHOLd:STIMe * |
| :VIOLation:TRANsition:GTHan | :VIOLation:TRANsition:GTHan * |
| :VIOLation:TRANsition:LTHan | :VIOLation:TRANsition:LTHan * |
| :VIOLation:TRANsition:SOURce:HTHReshold | :VIOLation:TRANsition:SOURce:HTHReshold * |
| :VIOLation:TRANsition:SOURce:LEVel | :VIOLation:TRANsition:SOURce:LEVel * |
| :VIOLation:TRANsition:SOURce:LTHReshold | :VIOLation:TRANsition:SOURce:LTHReshold * |
| :VIOLation:TRANsition:TYPE | :VIOLation:TRANsition:TYPE * |
| Note | :TRIGger:DEVents:ARM:SLOPe |
| Note | :TRIGger:DEVents:ARM:SOURce |
| Note | :TRIGger:DEVents:EVENt:DELay |
| Note | :TRIGger:DEVents:EVENt:SLOPe |
| Note | :TRIGger:DEVents:EVENt:SOURce |
| Note | :TRIGger:DEVents:TRIGger:SLOPe |
| Note | :TRIGger:DEVents:TRIGger:SOURce |
| Note | :TRIGger:DTIMe:ARM:SLOPe |
| Note | :TRIGger:DTIMe:ARM:SOURce |
| Note | :TRIGger:DTIMe:DELay |
| Note | :TRIGger:DTIMe:TRIGger:SLOPe |
| Note | :TRIGger:DTIMe:TRIGger:SOURce |
| :TRIGger:EDGE:COUPling | Note |
| :TRIGger:EDGE:SLOPe | Note |
| :TRIGger:EDGE:SOURce | Note |
| :TRIGger:GLITch:POLarity | :TRIGger:GLITch:POLarity |
| :TRIGger:GLITch:SOURce | :TRIGger:GLITch:SOURce |
| :TRIGger:GLITch:WIDTh | :TRIGger:GLITch:WIDTh |
| :TRIGger:HOLDoff | :TRIGger:HOLDoff |
| :TRIGger:HTHReshold | :TRIGger:HTHReshold * |
| :TRIGger:HTHReshold:LIMits? | :TRIGger:HTHReshold:LIMits? * |
| :TRIGger:HYSTeresis | :TRIGger:HYSTeresis |
| :TRIGger:LEVel | :TRIGger:LEVel |
| :TRIGger:LEVel:LIMits? | :TRIGger:LEVel:LIMits? * |
| :TRIGger:LTHReshold | :TRIGger:LTHReshold * |
| :TRIGger:MODE | :TRIGger:MODE |
| Note | :TRIGger:PATTern:CONDition |

| | |
|---|---|
| Note | :TRIGger:PATTern:LOGic |
| Note | :TRIGger:SLOPe |
| Note | :TRIGger:SOURce |
| Note | :TRIGger:STATe:CLOCk |
| Note | :TRIGger:STATe:CONDition |
| Note | :TRIGger:STATe:LOGic |
| Note | :TRIGger:STATe:SLOPe |
| Note | :TRIGger:STV:FIELd |
| Note | :TRIGger:STV:LINE |
| Note | :TRIGger:STV:SOURce |
| Note | :TRIGger:STV:SPOLarity |
| Note | :TRIGger:STV:STANdard |
| :TRIGger:SWEep | :TRIGger:SWEep |
| Note | :TRIGger:TDLY:ARM:SLOPe * |
| Note | :TRIGger:TDLY:ARM:SOURce * |
| Note | :TRIGger:TDLY:DELay * |
| Note | :TRIGger:TDLY:TRIGger:SLOPe * |
| Note | :TRIGger:TDLY:TRIGger:SOURce * |
| Note | :TRIGger:UDTV:ENUMber |
| Note | :TRIGger:UDTV:PGTHan |
| Note | :TRIGger:UDTV:PLTHan |
| Note | :TRIGger:UDTV:SLOPe |
| Note | :TRIGger:UDTV:SOURce |
| Note | :TRIGger:UDTV:STATe |
| Note | :TRIGger:BWLimit * |
| Note | :TRIGger:PROBe * |

# Waveform Command Language Compatibility

**HP 548XX and HP 547XX WAVeform Commands**

| HP 548XX | HP 547XX |
|---|---|
| :WAVeform:BANDpass? | :WAVeform:BANDpass? |
| :WAVeform:BYTeorder | :WAVeform:BYTeorder |
| :WAVeform:CLIPped? | :WAVeform:CLIPped? * |
| :WAVeform:COMPlete? | :WAVeform:COMPlete? |
| :WAVeform:COUNt? | :WAVeform:COUNt? |
| :WAVeform:COUPling? | :WAVeform:COUPling? |
| :WAVeform:DATA | :WAVeform:DATA |
| :WAVeform:FORMat | :WAVeform:FORMat |
| :WAVeform:POINts? | :WAVeform:POINts? |
| :WAVeform:PREamble | :WAVeform:PREamble |
| :WAVeform:SOURce | :WAVeform:SOURce |
| :WAVeform:TYPE? | :WAVeform:TYPE? |
| :WAVeform:VIEW | :WAVeform:VIEW |
| :WAVeform:XDISplay? | :WAVeform:XDISplay? |
| :WAVeform:XINCrement? | :WAVeform:XINCrement? |
| :WAVeform:XORigin? | :WAVeform:XORigin? |
| :WAVeform:XRANge? | :WAVeform:XRANge? |
| :WAVeform:XREFerence? | :WAVeform:XREFerence? |
| :WAVeform:XUNits? | :WAVeform:XUNits? |
| :WAVeform:YDISplay? | :WAVeform:YDISplay? |
| :WAVeform:YINCrement? | :WAVeform:YINCrement? |
| :WAVeform:YORigin? | :WAVeform:YORigin? |
| :WAVeform:YRANge? | :WAVeform:YRANge? |
| :WAVeform:YREFerence? | :WAVeform:YREFerence? |
| :WAVeform:YUNits? | :WAVeform:YUNits? |

# Waveform Memory Command Language Compatibility

### HP 548XX and HP 547XX Waveform MEMory Commands

| HP 548XX | HP 547XX |
|---|---|
| :WMEMory<N>:DISPlay | :WMEMory<N>:DISPlay |
| :WMEMory<N>:LoadFromFile | :WMEMory<N>:LoadFromFile * |
| :WMEMory<N>:SAVE | :WMEMory<N>:SAVE |
| :WMEMory<N>:XOFFset | :WMEMory<N>:XOFFset |
| :WMEMory<N>:XRANge | :WMEMory<N>:XRANge |
| :WMEMory<N>:YOFFset | :WMEMory<N>:YOFFset |
| :WMEMory<N>:YRANge | :WMEMory<N>:YRANge |

Note: Command not supported on this oscilloscope.

# Root Command Language Compatibility

### HP 548XX and HP 547XX Root Commands

| HP 548XX | HP 547XX |
|---|---|
| :AER? | :AER? |
| :AUToscale | :AUToscale |
| :BLANk | :BLANk |
| :CDISplay | :CDISplay |
| :DIGitize | :DIGitize |
| :MODel? | :MODel? |
| :OPEEnable | :OPEEnable |
| :OPERegister? | :OPERegister? |
| :OVLEnable (54846A/45A/35A only) | :OVLEnable * (54846A/45A/35A only) |
| :OVLRegister? (54846A/45A/35A only) | :OVLRegister? * (54846A/45A/35A only) |
| :PRINt | :PRINt |
| :RECall:SETup | :RECall:SETup |
| :RUN | :RUN |
| :SERial | :SERial |
| :SINGle | :SINGle |
| :STOP | :STOP |
| :STORe:SETup | :STORe:SETup |
| :STORe:WAVeform | :STORe:WAVeform |
| :TERegister? | :TERegister? |
| :VIEW | :VIEW |

# Common Command Language Compatibility

### HP 548XX and HP 547XX Common Commands

| HP 548XX | HP 547XX |
|---|---|
| *CLS | *CLS |
| *ESE | *ESE |
| *ESR? | *ESR? |
| *IDN? | *IDN? |
| *LRN? | *LRN? |
| *OPC | *OPC |
| *OPT? | *OPT? |
| *RCL | *RCL |
| *RST | *RST |
| *SAV | *SAV |
| *SRE | *SRE |
| *STB? | *STB? |
| *TRG | *TRG |
| *TST? | *TST? |
| *WAI | *WAI |

# Infiniium and HP 545XX Oscilloscopes Language Compatibility

# Infiniium and HP 545XX Oscilloscopes Language Compatibility

When developing new programs, you should use the Infiniium (HP 548XX) command language, as documented in the chapters in this manual. When using existing programs (that you do not want to modify) with Infiniium Oscilloscopes, the HP 547XX and HP 545XX command sets are provided as built-in languages. See "Some HP 545XX Commands are New" on the next page.

- This chapter describes language compatibility with HP 545XX oscilloscopes.
- Chapter 25 describes language compatibility with HP 547XX oscilloscopes.

The built-in command languages make your use of the Infiniium Oscilloscopes compatible with previously designed oscilloscopes — ones that you may already be used to operating. Being able to choose another command language is beneficial if you want to use existing programs on Infiniium Oscilloscopes without having to modify your programs.

The built-in HP 547XX and HP 545XX languages can also be helpful if you are familiar with one or both of them, and want to continue using that language on Infiniium Oscilloscopes.

### Selecting a Command Language

Use the :SYSTem:LANGuage command to select either the HP 548XX, HP 547XX, or HP 545XX command language built into the Infiniium Oscilloscopes. The HP 548XX command language is the default.

### Command Language Tables

There are some differences between the built-in command languages. The tables in this chapter show these differences (if any exist), and the relationships between the command languages for HP 548XX Infiniium Oscilloscopes and HP 545XX Oscilloscopes. If a command is supported only on the 54846S, 54845A, and 54835A, it is noted.

**Some HP 545XX Commands are New**

Some HP 545XX commands are new. That is, they were not in the original command language, but they have been added to the HP 545XX language support for HP 548XX Oscilloscopes.

**What the * Symbol Means**

In the command tables, the " * " symbol after a command indicates the command is newly supported.

**See Also**

See Chapter 25 for language compatability between Infiniium and HP 547XX Oscilloscopes.

## To select a command language

In Infiniium Oscilloscopes, you can select one of the supported command languages either over the GPIB or from the oscilloscope front panel.

- To select one of the command languages over the GPIB, enter the appropriate command for the oscilloscope type you are using:

  :SYSTem:LANGuage HP545XX

  :SYSTem:LANGuage HP548XX

  (This chapter describes the HP 545XX and HP 548XX command language compatibility.)

  :SYSTem:LANGuage HP547XX

  (Use the information in Chapter 25 for the HP 547XX Digitizing Oscilloscopes command language compatability.)

- To select one of the command languages from the oscilloscope front panel, select Utilities, Remote Interface, and Select Language. Then choose HP545XX, HP547XX, or HP548XX.

# Acquisition System Command Language Compatibility

### HP 548XX and HP 545XX ACQuire Commands

| HP 548XX | HP 545XX |
|---|---|
| :ACQuire:AVERage | Note |
| :ACQuire:AVERage:COUNt | Note |
| :ACQuire:BWLimit | :ACQuire:BWLimit * |
| :ACQuire:COMPlete | :ACQuire:COMPlete (COMPlete:STATe is on when this feature is selected) |
| :ACQuire:COMPlete:STATe | :ACQuire:COMPlete:STATe * |
| :ACQuire:CONFig (54846A/45A/35A only) | :ACQuire:CONFig * (54846A/45A/35A only) |
| :ACQuire:COUNt | :ACQuire:COUNt |
| :ACQuire:INTerpolate | :ACQuire:INTerpolate * |
| :ACQuire:MODE | Note |
| :ACQuire:POINts | :ACQuire:POINts |
| :ACQuire:SRATe | Note |
| Note | :ACQuire:TYPE |

Note: Command not supported on this oscilloscope.

# Calibration Command Language Compatibility

### HP 548XX and HP 545XX CALibrate Commands

| HP 548XX | HP 545XX |
|---|---|
| :CALibrate:CANCel | Note |
| :CALibrate:CONTinue | Note |
| Note | :CALibrate:DATA:ASCii? |
| :CALibrate:MPRotect | Note |
| :CALibrate:OUTPut | Note |
| :CALibrate:SDONe? | Note |
| :CALibrate:SKEW | Note |
| :CALibrate:STARt | Note |
| :CALibrate:STATus? | Note |
| Note | :CALibrate:TNULl <value> |

Note: Command not supported on this oscilloscope.

# Channel Command Language Compatibility

### HP 548XX and HP 545XX CHANnel Commands

| HP 548XX | HP 545XX |
|---|---|
| :CHANnel<N>:BWLimit | Note |
| Note | :CHANnel<N>:COUPling |
| :CHANnel<N>:DISPlay | :CHANnel<N>:DISPlay |
| :CHANnel<N>:ECL | :CHANnel<N>:ECL |
| Note | :CHANnel<N>:HFReject |
| :CHANnel<N>:INPut | :CHANnel<N>:INPut * |
| :CHANnel<N>:OFFSet | :CHANnel<N>:OFFSet |
| :CHANnel<N>:PROBe | :CHANnel<N>:PROBe |
| :CHANnel<N>:PROBe:EGAin | :CHANnel<N>:PROBe:EGAin * |
| :CHANnel<N>:PROBe:EOFFset | :CHANnel<N>:PROBe:EOFFset * |
| :CHANnel<N>:PROBe:SKEW | :CHANnel<N>:PROBe:SKEW * |
| :CHANnel<N>:PROTection? (54846A/45A/35A only) | Note |
| :CHANnel<N>:PROTection:CLEar (54846A/45A/35A only) | Note |
| :CHANnel<N>:RANGe | :CHANnel<N>:RANGe |
| Note | :CHANnel<N>:SETup? |
| :CHANnel<N>:TTL | :CHANnel<N>:TTL |
| :CHANnel<N>:UNITs | :CHANnel<N>:UNITs * |

# Disk Command Language Compatibility

### HP 548XX and HP 545XX DISK Commands

| HP 548XX | HP 545XX |
|---|---|
| :DISK:CDIRectory | :DISK:CDIRectory |
| :DISK:DELete | :DISK:DELete |
| :DISK:DIRectory? | :DISK:DIRectory? |
| :DISK:GetFILe? | :DISK:GetFILe? * |
| :DISK:LOAD | :DISK:LOAD |
| :DISK:MDIRectory | :DISK:MDIRectory |
| :DISK:PWD? | :DISK:PWD? |
| Note | :DISK:SIMage |
| :DISK:STORe | :DISK:STORe |

Note: Command not supported on this oscilloscope.

# Display Command Language Compatibility

### HP 548XX and HP 545XX DISPlay Commands

| HP 548XX | HP 545XX |
|---|---|
| :DISPlay:COLumn | :DISPlay:COLumn |
| :DISPlay:CONNect | :DISPlay:CONNect |
| :DISPlay:DATA? | :DISPlay:DATA |
| :DISPlay:DCOLor | :DISPlay:DCOLor * |
| Note | :DISPlay:FORMat |
| :DISPlay:GRATicule | :DISPlay:GRATicule |
| :DISPlay:GRATicule:INTensity | :DISPlay:GRATicule:INTensity * |
| Note | :DISPlay:INVerse |
| :DISPlay:LAYout:MBAR | :DISPlay:LAYout:MBAR * |
| :DISPlay:LAYout:MRESults | :DISPlay:LAYout:MRESults * |
| :DISPlay:LINE | :DISPlay:LINE |
| :DISPlay:PERSistence | :DISPlay:PERSistence |
| :DISPlay:PERSistence:TIME | Note |
| :DISPlay:ROW | :DISPlay:ROW |
| :DISPlay:SCOLor | :DISPlay:SCOLor * |
| Note | :DISPlay:SETup? |
| :DISPlay:SSAVer | :DISPlay:SSAVer * |
| :DISPlay:SSAVer:AAFTer | :DISPlay:SSAVer:AAFTer * |
| :DISPlay:STRing | :DISPlay:STRing |
| :DISPlay:TEXT | :DISPlay:TEXT |
| Note | :DISPlay:MARKer\|TMARker\|VMARker |

Note: Command not supported on this oscilloscope.

# External Command Language Compatibility

### HP 548XX and HP 545XX EXTernal Commands

| HP 548XX | HP 545XX |
|---|---|
| :EXTernal:BWLimit | Note |
| Note | :EXTernal:COUPling |
| Note | :EXTernal:HFReject |
| :EXTernal:INPut | Note |
| :EXTernal:PROBe | :EXTernal:PROBe |
| :EXTernal:PROBe:EGAin | :EXTernal:PROBe:EGAin * |
| :EXTernal:PROBe:EOFFset | :EXTernal:PROBe:EOFFset * |
| :EXTernal:PROBe:SKEW | :EXTernal:PROBe:SKEW * |
| :EXTernal:RANGe | :EXTernal:RANGe |
| Note | :EXTernal:SETup? |
| :EXTernal:UNITs | :EXTernal:UNITs * |

Note: Command not supported on this oscilloscope.

# FFT Command Language Compatibility

This subsystem is not implemented for HP 548XX or HP 545XX Oscillscopes.

# Function Command Language Compatibility

### HP 548XX and HP 545XX FUNCtion Commands

| HP 548XX | HP 545XX |
|---|---|
| :FUNCtion<N>? | :FUNCtion<N>? * |
| :FUNCtion<N>:ADD | :FUNCtion<N>:ADD |
| :FUNCtion<N>:DIFF | :FUNCtion<N>:DIFF |
| :FUNCtion<N>:DISPlay | :FUNCtion<N>:DISPlay |
| :FUNCtion<N>:DIVide | :FUNCtion<N>:DIVide * |
| Note | :FUNCtion<N>:FFT |
| :FUNCtion<N>:FFT:FREQuency | :FUNCtion<N>:FREQuency |
| :FUNCtion<N>:FFT:RESolution? | Note |
| :FUNCtion<N>:FFT:WINDow | Note |
| :FUNCtion<N>:FFTMagnitude | Note |
| :FUNCtion<N>:HORizontal | :FUNCtion<N>:HORizontal |
| :FUNCtion<N>:HORizontal:POSition | :FUNCtion<N>:HORizontal:POSition |
| :FUNCtion<N>:HORizontal:RANGe | :FUNCtion<N>:HORizontal:RANGe |
| :FUNCtion<N>:INTegrate | :FUNCtion<N>:INTegrate |
| :FUNCtion<N>:INVert | :FUNCtion<N>:INVert |
| Note | :FUNCtion<N>:LEVel |
| :FUNCtion<N>:MAGNify | :FUNCtion<N>:MAGNify |
| :FUNCtion<N>:MAXimum | :FUNCtion<N>:MAXimum * |
| :FUNCtion<N>:MINimum | :FUNCtion<N>:MINimum * |
| Note | :FUNCtion<N>:MODE? |
| :FUNCtion<N>:MULTiply | :FUNCtion<N>:MULTiply |
| :FUNCtion<N>:OFFSet | :FUNCtion<N>:OFFSet |
| Note | :FUNCtion<N>:ONLY |
| Note | :FUNCtion<N>:PEAK |
| :FUNCtion<N>:RANGe | :FUNCtion<N>:RANGe |
| Note | :FUNCtion<N>:SETup? |
| :FUNCtion<N>:SUBTract | :FUNCtion<N>:SUBTract |
| :FUNCtion<N>:VERSus | :FUNCtion<N>:VERSus |
| :FUNCtion<N>:VERTical | :FUNCtion<N>:VERTical |
| :FUNCtion<N>:VERTical:OFFSet | :FUNCtion<N>:VERTical:OFFSet |
| :FUNCtion<N>:VERTical:RANGe | :FUNCtion<N>:VERTical:RANGe |
| Note | :FUNCtion<N>:WINDow |

Note: Command not supported on this oscilloscope.

# Hardcopy Command Language Compatibility

### HP 548XX and HP 545XX HARDcopy Commands

| HP 548XX | HP 545XX |
|---|---|
| :HARDcopy:AREA | :HARDcopy:AREA * |
| :HARDcopy:DPRinter | :HARDcopy:DPRinter * |
| :HARDcopy:FACTors | :HARDcopy:FACTors * |
| :HARDcopy:IMAGe | :HARDcopy:IMAGe * |
| :HARDcopy:PRINters? | :HARDcopy:PRINters? * |

# Limit Test Command Language Compatibility

Limit TESt commands do not apply to HP 548XX or HP 545XX Oscilloscopes.

# Marker Command Language Compatibility

### HP 548XX and HP 545XX MARKer Commands

| HP 548XX | HP 545XX |
|---|---|
| :MARKer:CURSor? | :MARKer:CURSor? * |
| Note | :MARKer:DISPlay |
| :MARKer:MEASurement:READout | :MARKer:MEASurement:READout * |
| :MARKer:MODE | :MARKer:MODE |
| Note | :MARKer:SETup? |
| :MARKer:TDELta? | :MARKer:TDELta? * |
| :MARKer:TSTArt | :MARKer:TSTArt * |
| :MARKer:TSTOp | :MARKer:TSTOp * |
| :MARKer:VDELta? | :MARKer:VDELta? * |
| :MARKer:VSTArt | :MARKer:VSTArt * |
| :MARKer:VSTOp | :MARKer:VSTOp * |
| :MARKer:X1Position | :MARKer:X1Position |
| :MARKer:X1Y1source | :MARKer:X1Y1source |
| :MARKer:X2Position | :MARKer:X2Position |
| :MARKer:X2Y2source | :MARKer:X2Y2source |
| :MARKer:XDELta? | :MARKer:XDELta? |
| :MARKer:Y1Position | :MARKer:Y1Position |
| :MARKer:Y2Position | :MARKer:Y2Position |
| :MARKer:YDELta? | :MARKer:YDELta? |

Note: Command not supported on this oscilloscope.

### Sources for MARKer Commands

Sources for the MARKer commands can be CHANnel<N>, FUNCtion<N>, or WMEMory<N>.

<N> CHANnel<N> is:

An integer, 1 - 2, for 54810/54820 Infiniium Oscilloscopes.
An integer, 1 - 4, for all other Infiniium Oscilloscope models.

FUNCtion<N> and WMEMory<N> are:

Integers, 1 - 4, representing the selected function or waveform memory.

# Measure Command Language Compatibility

### HP 548XX and HP 545XX MEASure Commands

| HP 548XX | HP 545XX |
|---|---|
| :MEASure:ALL? | :MEASure:ALL? |
| :MEASure:CLEar | :MEASure:CLEar * |
| :MEASure:DEFine | :MEASure:DEFine |
| :MEASure:DELTatime | :MEASure:DELTatime * |
| :MEASure:DUTYcycle | :MEASure:DUTYcycle |
| :MEASure:FALLtime | :MEASure:FALLtime |
| :MEASure:FFT:DFRequency | Note |
| :MEASure:FFT:DMAGnitude | Note |
| :MEASure:FFT:FREQuency | Note |
| :MEASure:FFT:MAGNitude | Note |
| :MEASure:FFT:PEAK1 | Note |
| :MEASure:FFT:PEAK2 | Note |
| :MEASure:FFT:THReshold | Note |
| :MEASure:FREQuency | :MEASure:FREQuency |
| :MEASure:NWIDth | :MEASure:NWIDth |
| :MEASure:OVERshoot | :MEASure:OVERshoot |
| :MEASure:PERiod | :MEASure:PERiod |
| :MEASure:PREShoot | :MEASure:PREShoot |
| :MEASure:PWIDth | :MEASure:PWIDth |
| :MEASure:RESults? | :MEASure:RESults? |
| :MEASure:RISetime | :MEASure:RISetime |
| :MEASure:SCRatch | :MEASure:SCRatch |
| :MEASure:SENDvalid | :MEASure:SENDvalid * |
| :MEASure:SOURce | :MEASure:SOURce |
| :MEASure:STATistics | :MEASure:STATistics |
| :MEASure:TEDGe | :MEASure:TEDGe * |
| :MEASure:TMAX | :MEASure:TMAX |
| :MEASure:TMIN | :MEASure:TMIN |
| Note | :MEASure:TSTArt |
| Note | :MEASure:TSTOp |
| :MEASure:TVOLt | :MEASure:TVOLt |
| Note | :MEASure:UNITs |
| Note | :MEASure:UPPer |
| Note | :MEASure:VACRms |
| :MEASure:VAMPlitude | :MEASure:VAMPlitude |
| :MEASure:VAVerage | :MEASure:VAVerage |
| :MEASure:VBASe | :MEASure:VBASe |
| Note | :MEASure:VDCRms |
| Note | :MEASure:VDELta? |
| Note | :MEASure:VFIFty |
| :MEASure:VLOWer | :MEASure:VLOWer * |
| :MEASure:VMAX | :MEASure:VMAX |
| :MEASure:VMIDdle | :MEASure:VMIDdle * |
| :MEASure:VMIN | :MEASure:VMIN |

| | |
|---|---|
| :MEASure:VPP | :MEASure:VPP |
| Note | :MEASure:VRELative |
| Note | :MEASure:VSTArt |
| Note | :MEASure:VSTOp |
| :MEASure:VRMS | :MEASure:VRMS * |
| :MEASure:VTIMe | :MEASure:VTIMe * |
| :MEASure:VTOP | :MEASure:VTOP |
| :MEASure:VUPPer | :MEASure:VUPPer * |

Note: Command not supported on this oscilloscope.

# Multiple Memory Command Language Compatibility

Multiple MEMory commands do not apply to HP 548XX Oscilloscopes.

# Memory Test Command Language Compatibility

Memory TESt commands do not apply to HP 548XX or HP 545XX Oscilloscopes.

# Pixel Memory Command Language Compatibility

Pixel MEMory commands do not apply to HP 548XX Oscilloscopes.

# Self-Test Command Language Compatibility

### HP 548XX and HP 545XX SELFtest Commands

| HP 548XX | HP 545XX |
|---|---|
| :SELFtest:ASET | :SELFtest:ASET * |
| :SELFtest:SCOPETEST | :SELFtest:SCOPETEST * |

# Sequential Command Language Compatibility

SEQuential commands do not apply to HP 548XX Oscilloscopes.

# System Command Language Compatibility

### HP 548XX and HP 545XX SYSTem Commands

| HP 548XX | HP 545XX |
|---|---|
| :SYSTem:DATE | :SYSTem:DATE |
| :SYSTem:DEBug | :SYSTem:DEBug * |
| :SYSTem:DSP | :SYSTem:DSP |
| :SYSTem:ERRor? | :SYSTem:ERRor? |
| :SYSTem:HEADer | :SYSTem:HEADer |
| :SYSTem:HELP:HEADers? | :SYSTem:HELP:HEADers? * |
| :SYSTem:HPIB:ADDRess? | :SYSTem:HPIB:ADDRess? * |
| :SYSTem:LANGuage | :SYSTem:LANGuage * |
| :SYSTem:LONGform | :SYSTem:LONGform |
| :SYSTem:SETup | :SYSTem:SETup |
| :SYSTem:TIME | :SYSTem:TIME |

# Time Base Command Language Compatibility

### HP 548XX and HP 545XX TIMebase Commands

| HP 548XX | HP 545XX |
|---|---|
| :TIMebase:DELay | :TIMebase:DELay |
| Note | :TIMebase:MODE |
| :TIMebase:POSition | :TIMebase:POSition * |
| :TIMebase:RANGe | :TIMebase:RANGe |
| :TIMebase:REFerence | :TIMebase:REFerence |
| Note | :TIMebase:RLENgth |
| Note | :TIMebase:SAMPle |
| Note | :TIMebase:SAMPle:CLK |
| Note | :TIMebase:SAMPle:CLOCk |
| :TIMebase:SCALe | Note |
| :TIMebase:SETup? | :TIMebase:SETup? |
| :TIMebase:VIEW | :TIMebase:VIEW * |
| :TIMebase:WINDow:DELay | :TIMebase:WINDow:DELay * |
| :TIMebase:WINDow:POSition | :TIMebase:WINDow:POSition * |
| :TIMebase:WINDow:RANGe | :TIMebase:WINDow:RANGe * |
| :TIMebase:WINDow:SCALe | Note |
| :TIMebase:WINDow:SOURce | :TIMebase:WINDow:SOURce * |

# Trigger Command Language Compatibility

### HP 548XX and HP 545XX TRIGger Commands

| HP 548XX | HP 545XX |
|---|---|
| :TRIGger:ADVanced | Note |
| :DELay:EDLY:ARM:SLOPe | Note |
| :DELay:EDLY:ARM:SOURce | Note |
| :DELay:EDLY:EVENt:DELay | Note |
| :DELay:EDLY:EVENt:SLOPe | Note |
| :DELay:EDLY:EVENt:SOURce | Note |
| :DELay:EDLY:TRIGger:SLOPe | Note |
| :DELay:EDLY:TRIGger:SOURce | Note |
| :DELay:MODE | Note |
| :DELay:TDLY:ARM:SLOPe | Note |
| :DELay:TDLY:ARM:SOURce | Note |
| :DELay:TDLY:DELay | Note |
| :DELay:TDLY:TRIGger:SLOPe | Note |
| :DELay:TDLY:TRIGger:SOURce | Note |
| :MODE | Note |
| :PATTern:CONDition | Note |
| :PATTern:LOGic | Note |
| :STATe:CLOCk | Note |
| :STATe:CONDition | Note |
| :STATe:LOGic | Note |
| :STATe:LTYPe | Note |
| :STATe:SLOPe | Note |
| :TV:MODE | Note |
| :TV:STV:FIELd | Note |
| :TV:STV:LINE | Note |
| :TV:STV:SOURce | Note |
| :TV:STV:SPOLarity | Note |
| :TV:STV:STANdard | Note |
| :TV:UDTV:EDGE | Note |
| :TV:UDTV:ENUMber | Note |
| :TV:UDTV:PGTHan | Note |
| :TV:UDTV:PLTHan | Note |
| :TV:UDTV:POLarity | Note |
| :TV:UDTV:SOURce | Note |
| :VIOLation:MODE | :VIOLation:MODE |
| :VIOLation:PWIDth:DIRection | :VIOLation:PWIDth:DIRection * |
| :VIOLation:PWIDth:POLarity | :VIOLation:PWIDth:POLarity * |
| :VIOLation:PWIDth:SOURce | :VIOLation:PWIDth:SOURce * |
| :VIOLation:PWIDth:WIDTh | :VIOLation:PWIDth:WIDTh * |
| :VIOLation:RUNT:DIRection | :VIOLation:RUNT:DIRection * |
| :VIOLation:RUNT:SOURce:HTHReshold | :VIOLation:RUNT:SOURce:HTHReshold * |
| :VIOLation:RUNT:SOURce:LTHReshold | :VIOLation:RUNT:SOURce:LTHReshold * |
| :VIOLation:RUNT:TQUalified | :VIOLation:RUNT:TQUalified * |
| :VIOLation:SETup:HOLD:CSOurce:EDGE | :VIOLation:SETup:HOLD:CSOurce:EDGE * |

| | |
|---|---|
| :VIOLation:SETup:HOLD:CSOurce:LEVel | :VIOLation:SETup:HOLD:CSOurce:LEVel * |
| :VIOLation:SETup:HOLD:DSOurce:HTHReshold | :VIOLation:SETup:HOLD:DSOurce:HTHReshold * |
| :VIOLation:SETup:HOLD:DSOurce:LTHReshold | :VIOLation:SETup:HOLD:DSOurce:LTHReshold * |
| :VIOLation:SETup:HOLD:TIME | :VIOLation:SETup:HOLD:TIME * |
| :VIOLation:SETup:MODE | :VIOLation:SETup:MODE * |
| :VIOLation:SETup:SETup:CSOurce:EDGE | :VIOLation:SETup:SETup:CSOurce:EDGE * |
| :VIOLation:SETup:SETup:CSOurce:LEVel | :VIOLation:SETup:SETup:CSOurce:LEVel * |
| :VIOLation:SETup:SETup:DSOurce:HTHReshold | :VIOLation:SETup:SETup:DSOurce:HTHReshold * |
| :VIOLation:SETup:SETup:DSOurce:LTHReshold | :VIOLation:SETup:SETup:DSOurce:LTHReshold * |
| :VIOLation:SETup:SETup:TIME | :VIOLation:SETup:SETup:TIME * |
| :VIOLation:SETup:SHOLd:CSOurce:EDGE | :VIOLation:SETup:SHOLd:CSOurce:EDGE * |
| :VIOLation:SETup:SHOLd:CSOurce:LEVel | :VIOLation:SETup:SHOLd:CSOurce:LEVel * |
| :VIOLation:SETup:SHOLd:DSOurce:HTHReshold | :VIOLation:SETup:SHOLd:DSOurce:HTHReshold * |
| :VIOLation:SETup:SHOLd:DSOurce:LTHReshold | :VIOLation:SETup:SHOLd:DSOurce:LTHReshold * |
| :VIOLation:SETup:SHOLd:HOLDTIME | :VIOLation:SETup:SHOLd:HOLDTIME * |
| :VIOLation:SETup:SHOLd:HTIMe | :VIOLation:SETup:SHOLd:HTIMe * |
| :VIOLation:SETup:SHOLd:SETUPTIME | :VIOLation:SETup:SHOLd:SETUPTIME * |
| :VIOLation:SETup:SHOLd:STIMe | :VIOLation:SETup:SHOLd:STIMe * |
| :VIOLation:TRANsition:GTHan | :VIOLation:TRANsition:GTHan * |
| :VIOLation:TRANsition:LTHan | :VIOLation:TRANsition:LTHan * |
| :VIOLation:TRANsition:SOURce:HTHReshold | :VIOLation:TRANsition:SOURce:HTHReshold * |
| :VIOLation:TRANsition:SOURce:LEVel | :VIOLation:TRANsition:SOURce:LEVel * |
| :VIOLation:TRANsition:SOURce:LTHReshold | :VIOLation:TRANsition:SOURce:LTHReshold * |
| :VIOLation:TRANsition:TYPE | :VIOLation:TRANsition:TYPE * |
| Note | :TRIGger:CENTered |
| Note | :TRIGger:CONDition |
| Note | :TRIGger:COUPling |
| Note | :TRIGger:DELay |
| Note | :TRIGger:DELay:SLOPe |
| Note | :TRIGger:DELay:SOURce |
| :TRIGger:EDGE:COUPling | Note |
| :TRIGger:EDGE:SLOPe | Note |
| :TRIGger:EDGE:SOURce | Note |
| Note | :TRIGger:FIELd |
| Note | :TRIGger:GLITch:CENTered |
| Note | :TRIGger:GLITch:HOLDoff |
| Note | :TRIGger:GLITch:LEVel |
| :TRIGger:GLITch:POLarity | :TRIGger:GLITch:POLarity * |
| :TRIGger:GLITch:SOURce | :TRIGger:GLITch:SOURce |
| :TRIGger:GLITch:WIDTh | :TRIGger:GLITch:WIDTh |
| :TRIGger:HOLDoff | :TRIGger:HOLDoff |
| Note | :TRIGger:LEVel |
| :TRIGger:HTHReshold | Note |
| :TRIGger:HTHReshold:LIMits? | Note |
| :TRIGger:HYSTeresis | Note |
| :TRIGger:LEVel | Note |
| :TRIGger:LEVel:LIMits? | Note |

| | |
|---|---|
| Note | :TRIGger:LINE |
| Note | :TRIGger:LOGic |
| :TRIGger:LTHReshold | :TRIGger:LTHReshold * |
| :TRIGger:MODE | :TRIGger:MODE |
| Note | :TRIGger:NREJect |
| Note | :TRIGger:OCCurrence |
| Note | :TRIGger:OCCurrence:SLOPe |
| Note | :TRIGger:OCCurrence:SOURce |
| Note | :TRIGger:PATH |
| Note | :TRIGger:POLarity |
| Note | :TRIGger:QUALify |
| Note | :TRIGger:SETup? |
| Note | :TRIGger:SLOPe |
| Note | :TRIGger:SOURce |
| Note | :TRIGger:STANdard |
| :TRIGger:SWEep | |

# Waveform Command Language Compatibility

### HP 548XX and HP 545XX WAVeform Commands

| HP 548XX | HP 545XX |
|---|---|
| :WAVeform:BANDpass? | Note |
| :WAVeform:BYTeorder | Note |
| :WAVeform:CLIPped? | :WAVeform:CLIPped? * |
| :WAVeform:COMPlete? | :WAVeform:COMPlete? |
| :WAVeform:COUNt? | :WAVeform:COUNt? |
| :WAVeform:COUPling? | Note |
| :WAVeform:DATA | :WAVeform:DATA |
| :WAVeform:FORMat | :WAVeform:FORMat |
| :WAVeform:POINts? | :WAVeform:POINts? |
| :WAVeform:PREamble | :WAVeform:PREamble |
| :WAVeform:SOURce | :WAVeform:SOURce |
| :WAVeform:TYPE? | :WAVeform:TYPE? |
| :WAVeform:VIEW | Note |
| :WAVeform:XDISplay? | Note |
| :WAVeform:XINCrement? | :WAVeform:XINCrement? |
| :WAVeform:XORigin? | :WAVeform:XORigin? |
| :WAVeform:XRANge? | Note |
| :WAVeform:XREFerence? | :WAVeform:XREFerence? |
| :WAVeform:XUNits? | Note |
| :WAVeform:YDISplay? | Note |
| :WAVeform:YINCrement? | :WAVeform:YINCrement? |
| :WAVeform:YORigin? | :WAVeform:YORigin? |
| :WAVeform:YRANge? | Note |
| :WAVeform:YREFerence? | :WAVeform:YREFerence? |
| :WAVeform:YUNits? | Note |

# Waveform Memory Command Language Compatibility

### HP 548XX and HP 545XX Waveform MEMory Commands

| HP 548XX | HP 545XX |
|---|---|
| :WMEMory<N>:DISPlay | :WMEMory<N>:DISPlay |
| :WMEMory<N>:LoadFromFile | :WMEMory<N>:LoadFromFile * |
| :WMEMory<N>:SAVE | :WMEMory<N>:SAVE * |
| :WMEMory<N>:XOFFset | :WMEMory<N>:XOFFset |
| :WMEMory<N>:XRANge | :WMEMory<N>:XRANge |
| :WMEMory<N>:YOFFset | :WMEMory<N>:YOFFset |
| :WMEMory<N>:YRANge | :WMEMory<N>:YRANge |

# Root Command Language Compatibility

### HP 548XX and HP 545XX Root Commands

| HP 548XX | HP 545XX |
|---|---|
| :AER? | :AER? * |
| :AUToscale | :AUToscale |
| :BLANk | :BLANk |
| :CDISplay | :CDISplay * |
| :DIGitize | :DIGitize |
| :MODel? | :MODel? * |
| :OPEEnable | :OPEEnable * |
| :OPERegister? | :OPERegister? * |
| :OVLEnable (54846A/45A/35A only) | :OVLEnable * (54846A/45A/35A only) |
| :OVLRegister? (54846A/45A/35A only) | :OVLRegister? * (54846A/45A/35A only) |
| :PRINt | :PRINt |
| :RECall:SETup | :RECall:SETup * |
| :RUN | :RUN |
| :SERial | :SERial |
| :SINGle | :SINGle * |
| Note | :STATus? |
| :STOP | :STOP |
| :STORe:SETup | Note |
| :STORe:WAVeform | Note |
| Note | :STORe |
| :TERegister? | :TERegister? |
| :VIEW | :VIEW |

# Common Command Language Compatibility

### HP 548XX and HP 545XX Common Commands

| HP 548XX | HP 545XX |
|---|---|
| *CLS | *CLS |
| *ESE | *ESE |
| *ESR? | *ESR? |
| *IDN? | *IDN? |
| *LRN? | *LRN? |
| *OPC | *OPC |
| *OPT? | *OPT? |
| *RCL | *RCL |
| *RST | *RST |
| *SAV | *SAV |
| *SRE | *SRE |
| *STB? | *STB? |
| *TRG | *TRG |
| *TST? | *TST? |
| *WAI | *WAI |

29

# Error Messages

# Error Messages

This chapter describes the error messages and how they are generated. The possible causes for the generation of the error messages are also listed in the following table.

## Error Queue

As errors are detected, they are placed in an error queue. This queue is first in, first out. If the error queue overflows, the last error in the queue is replaced with error -350, "Queue overflow." Anytime the error queue overflows, the oldest errors remain in the queue, and the most recent error is discarded. The length of the oscilloscope's error queue is 30 (29 positions for the error messages, and 1 position for the "Queue overflow" message). Reading an error from the head of the queue removes that error from the queue, and opens a position at the tail of the queue for a new error. When all errors have been read from the queue, subsequent error queries return 0, "No error."

The error queue is cleared when any of the following occur:

- the instrument is powered up,
- a *CLS command is sent,
- the last item from the queue is read, or
- the instrument is switched from talk only to addressed mode on the front panel.

## Error Numbers

The error numbers are grouped according to the type of error that is detected.

- +0 indicates no errors were detected.
- -100 to -199 indicates a command error was detected
- -200 to -299 indicates an execution error was detected.
- -300 to -399 indicates a device-specific error was detected.
- -400 to -499 indicates a query error was detected.
- +1 to +32767 indicates an oscilloscope specific error has been detected.

## Command Error

An error number in the range -100 to -199 indicates that an IEEE 488.2 syntax error has been detected by the instrument's parser. The occurrence of any error in this class sets the command error bit (bit 5) in the event status register and indicates that one of the following events occurred:

- An IEEE 488.2 syntax error was detected by the parser. That is, a computer-to-oscilloscope message was received that is in violation of the IEEE 488.2 standard. This may be a data element that violates the oscilloscope's listening formats, or a data type that is unacceptable to the oscilloscope.
- An unrecognized header was received. Unrecognized headers include incorrect oscilloscope-specific headers and incorrect or unimplemented IEEE 488.2 common commands.
- A Group Execute Trigger (GET) was entered into the input buffer inside of an IEEE 488.2 program message.

Events that generate command errors do not generate execution errors, oscilloscope-specific errors, or query errors.

## Execution Error

An error number in the range -200 to -299 indicates that an error was detected by the instrument's execution control block. The occurrence of any error in this class causes the execution error bit (bit 4) in the event status register to be set. It also indicates that one of the following events occurred:

- The program data following a header is outside the legal input range or is inconsistent with the oscilloscope's capabilities.
- A valid program message could not be properly executed due to some oscilloscope condition.

Execution errors are reported by the oscilloscope after expressions are evaluated and rounding operations are completed. For example, rounding a numeric data element will not be reported as an execution error. Events that generate execution errors do not generate command errors, oscilloscope specific errors, or query errors.

## Device- or Oscilloscope-Specific Error

An error number in the range of -300 to -399 or +1 to +32767 indicates that the instrument has detected an error caused by an oscilloscope operation that did not properly complete. This may be due to an abnormal hardware or firmware condition. For example, this error may be generated by a self-test response error, or a full error queue. The occurrence of any error in this class causes the oscilloscope-specific error bit (bit 3) in the event status register to be set.

## Query Error

An error number in the range -400 to -499 indicates that the output queue control of the instrument has detected a problem with the message exchange protocol. An occurrence of any error in this class should cause the query error bit (bit 2) in the event status register to be set. An occurrence of an error also means one of the following is true:

- An attempt is being made to read data from the output queue when no output is either present or pending.
- Data in the output queue has been lost.

## List of Error Messages

Figure 29-1 is a list of the error messages that are returned by the parser on this oscilloscope.

**Figure 29-1**

**Error Messages**

| | | |
|---|---|---|
| 0 | No error | The error queue is empty. Every error in the queue has been read (SYSTEM:ERROR? query) or the queue was cleared by power-up or *CLS. |
| -100 | Command error | This is the generic syntax error used if the oscilloscope cannot detect more specific errors. |
| -101 | Invalid character | A syntactic element contains a character that is invalid for that type. |
| -102 | Syntax error | An unrecognized command or data type was encountered. |
| -103 | Invalid separator | The parser was expecting a separator and encountered an illegal character. |
| -104 | Data type error | The parser recognized a data element different than one allowed. For example, numeric or string data was expected but block data was received. |
| -105 | GET not allowed | A Group Execute Trigger was received within a program message. |
| -108 | Parameter not allowed | More parameters were received than expected for the header. |
| -109 | Missing parameter | Fewer parameters were received than required for the header. |
| -112 | Program mnemonic too long | The header or character data element contains more than twelve characters. |
| -113 | Undefined header | The header is syntactically correct, but it is undefined for the oscilloscope. For example, *XYZ is not defined for the oscilloscope. |
| -121 | Invalid character in number | An invalid character for the data type being parsed was encountered. For example, a "9" in octal data. |
| -123 | Numeric overflow | Number is too large or too small to be represented internally. |
| -124 | Too many digits | The mantissa of a decimal numeric data element contained more than 255 digits excluding leading zeros. |
| -128 | Numeric data not allowed | A legal numeric data element was received, but the oscilloscope does not accept one in this position for the header. |
| -131 | Invalid suffix | The suffix does not follow the syntax described in IEEE 488.2 or the suffix is inappropriate for the oscilloscope. |
| -138 | Suffix not allowed | A suffix was encountered after a numeric element that does not allow suffixes. |
| -141 | Invalid character data | Either the character data element contains an invalid character or the particular element received is not valid for the header. |
| -144 | Character data too long | |
| -148 | Character data not allowed | A legal character data element was encountered where prohibited by the oscilloscope. |
| -150 | String data error | This error can be generated when parsing a string data element. This particular error message is used if the oscilloscope cannot detect a more specific error. |
| -151 | Invalid string data | A string data element was expected, but was invalid for some reason. For example, an END message was received before the terminal quote character. |

| | | |
|---|---|---|
| -158 | String data not allowed | A string data element was encountered but was not allowed by the oscilloscope at this point in parsing. |
| -160 | Block data error | This error can be generated when parsing a block data element. This particular error message is used if the oscilloscope cannot detect a more specific error. |
| -161 | Invalid block data | |
| -168 | Block data not allowed | A legal block data element was encountered but was not allowed by the oscilloscope at this point in parsing. |
| -170 | Expression error | This error can be generated when parsing an expression data element. It is used if the oscilloscope cannot detect a more specific error. |
| -171 | Invalid expression | |
| -178 | Expression data not allowed | Expression data was encountered but was not allowed by the oscilloscope at this point in parsing. |
| -200 | Execution error | This is a generic syntax error which is used if the oscilloscope cannot detect more specific errors. |
| -212 | Arm ignored | |
| -213 | Init ignored | |
| -214 | Trigger deadlock | |
| -215 | Arm deadlock | |
| -220 | Parameter error | |
| -221 | Settings conflict | |
| -222 | Data out of range | Indicates that a legal program data element was parsed but could not be executed because the interpreted value is outside the legal range defined by the oscilloscope. |
| -223 | Too much data | Indicates that a legal program data element of block, expression, or string type was received that contained more data than the oscilloscope could handle due to memory or related oscilloscope-specific requirements. |
| -224 | Illegal parameter value | |
| -230 | Data corrupt or stale | |
| -231 | Data questionable | |
| -240 | Hardware error | |
| -241 | Hardware missing | |
| -250 | Mass storage error | |
| -251 | Missing mass storage | |
| -252 | Missing media | |
| -253 | Corrupt media | |
| -254 | Media full | |
| -255 | Directory full | |
| -256 | File name not found | |
| -257 | File name error | |
| -258 | Media protected | |
| -260 | Expression error | |

| | | |
|---|---|---|
| -261 | Math error in expression | |
| -300 | Device specific error | |
| -310 | System error | Indicates that a system error occurred. |
| -311 | Memory error | |
| -312 | PUD memory error | |
| -313 | Calibration memory lost | |
| -314 | Save/recall memory lost | |
| -315 | Configuration memory lost | |
| -321 | Out of memory | |
| -330 | Self-test failed | |
| -350 | Queue overflow | Indicates that there is no room in the error queue and an error occurred but was not recorded. |
| -370 | No sub tests are defined for the selected self test | |
| -371 | Self Test status is corrupt or no self test has been executed | |
| -372 | This product configuration does not support the requested self test | |
| -373 | This product configuration does not support the requested source | |
| -374 | The requested self test log file could not be found | |
| -375 | Attenuator relay actuation counts can only be modified during factory service | |
| -400 | Query error | This is the generic query error. |
| -410 | Query INTERRUPTED | |
| -420 | Query UNTERMINATED | |
| -430 | Query DEADLOCKED | |
| -440 | Query UNTERMINATED after indefinite response | |

# Index


**Symbols**

...
Ellipsis 1-5

**Numerics**

707 1-19

**A**

Aborting a digitize operation 2-9
aborting a digitize operation 1-17
absolute voltage
and VMAX 20-92
and VMIN 20-95
accuracy and probe calibration 11-4
Acquire Commands 10-2
AllowMaxSR 10-3
AVERage 10-4
BWLimit 10-6
COMPlete 10-7
COMPlete STATe 10-9
CONFig 10-10
COUNt 10-5
INTerpolate 10-11
MODE 10-12
POINts 10-13
POINts AUTO 10-15
SRATe 10-16
SRATe AUTO 10-18
acquired data flow 5-3
acquisition
ACQuire AVER and completion 10-7
points 10-13
record length 10-13
sample program 6-7
sample rate 10-16
active probes and calibration 11-4
ADD 16-5
address, GPIB default 2-7
advanced
COMM triggering 24-25
delay trigger modes 24-43, 24-52
delay triggering 24-44, 24-53
logic triggering 24-32, 24-36
pattern triggering 24-33
state triggering 24-37
TV commands 24-59, 24-65
advanced trigger violation modes 24-75
pulse width violation mode 24-77
setup violation mode 24-83
transition violation mode 24-109
advisory line, reading and writing to 9-2
AER? 8-3
algebraic sum of functions 16-5
ALIGn 21-4
AlignFIT 21-5
ALL, and VIEW 25-44
AllowMaxSR 10-3
alphanumeric
characters in embedded string 1-12
strings 1-10
AMPS as vertical units 12-23, 15-17
AREA 17-3, 20-6
Arm Event Register
ARM bit 7-21
Arming the trigger 2-9
ASCII
and FORMat 25-32
character 32 1-5
linefeed 1-12
AttenSET?
in self-test commands 22-3
attenuation factor for probe 11-4, 12-7, 15-5
AUTO 10-18, 21-15
automatic measurements
sample program 6-8
AUToscale 8-4
during initialization 1-14
in sample program 6-17
availability of measured data 4-2
AVERage 10-4, 16-6, 21-16
and acquisition completion 10-7
and count 10-5, 21-17
AXIS 18-4

**B**

BANDpass query 25-5
bandwidth limit 25-5
filter 10-6
basic command structure 1-15
basic operations 1-2
BASIC sample programs 6-2
BIND
in MTESt SCALe command 21-34
Bit Definitions in Status Reporting 4-3
BLANk 8-5
and VIEW 8-24
blanking the user text area 14-21
block data 1-4, 1-20
and DATA 25-11
in learnstring 1-4
Block Diagram
Status Reporting Overview 4-3
Braces 1-5
Brackets
Square 1-5
buffer, output 1-9, 1-18
buffered responses 5-13
Bus Activity, Halting 2-9
Bus Commands 2-9
BWIDth
in TRIG ADV COMM 24-26
BWLimit 10-6, 12-3, 15-3
BYTE
and FORMat 25-33
Understanding the format 25-28
BYTeorder 25-6
and DATA 25-13

**C**

C sample programs 6-2
Calibration Commands 11-2, 11-5
CANCel 11-6
CONTinue 11-7
MPRotect 11-8
OUTPut 11-9
SDONe? 11-10
SKEW 11-11
STARt 11-12
STATus? 11-13
calibration status 11-13
CANCel 11-6
in self-test command 22-4
CDIRectory 13-3
CDISplay (Clear DISplay) 8-6
center screen voltage 12-6
CGRade 14-3
Channel Commands 12-2
BWLimit 12-3
DISPlay 12-4
EADapter 12-10
ECoupling 12-12
INPut 12-5
OFFSet 12-6

PROBe 12-7
PROBe ATTenuation 12-9
PROBe EGAin 12-14
PROBe EOFFset 12-15
PROBe GAIN 12-16, 15-13
PROBe ID? 12-17
PROBe PROTection CLEar 12-19
PROBe PROTection? 12-20
PROBe SKEW 12-18
RANGe 12-21
SCALe 12-22
UNITs 12-23
CHANnel PROBe ID? 12-17
channels, and VIEW 25-44
channel-to-channel skew factor 11-11
character program data 1-10
CLEar 20-13
Clearing
Buffers 2-9
Pending Commands 2-9
clearing
error queue 4-17, 29-3
registers and queues 4-18
Standard Event Status Register 4-11, 7-7
status data structures 7-4
TRG bit 4-10, 4-17
CLIPped query 25-7
clipped waveforms, and measurement error 20-5
CLOCk
and STATe 24-38
in TRIG ADV STATe 24-38
*CLS (Clear Status) 7-4
CME bit 7-6, 7-8
COLumn 14-7
combining
commands in same subsystem 1-7
long- and short-form headers 1-10
combining compound and simple commands 1-13
Command

EADapter 12-10, 15-7
ECoupling 12-12, 15-9
*ESE 7-5
ADD 16-5
AER? 8-3
ALIGn 21-4
AlignFIT 21-5
AMASk CREate 21-7
AMASk SAVE\STORe 21-9
AMASk SOURce 21-8
AMASk UNITs 21-10
AMASk XDELta 21-11
AMASk YDELta 21-13
AREA 17-3, 20-6
AttenSET? 22-3
AUTO 21-15
AUToscale 8-4
AVERage 10-3, 10-4, 16-6, 21-16
AVERage COUNt 21-17
AXIS 18-4
BLANk 8-5
BWLimit 10-6, 12-3, 15-3
CANCel 11-6, 22-4
CDIRectory 13-3
CDISplay 8-6
CGRade 14-3
LEVels? 14-5
CGRade CROSsing 20-7
CGRade DCDistortion 20-8
CGRade EHEight 20-9
CGRade EWIDth 20-10
CGRade JITTer 20-11
CGRade QFACtor 20-12
CHANnel PROBe ID? 12-17
CLEar 20-13
CLear Status 7-4
COLumn 14-7
COMPlete 10-7
COMPlete STATe 10-9
CONFig 10-10
CONNect 14-8
CONTinue 11-7
COUNt 10-5
COUNt FAILures? 21-18
COUNt FWAVeforms? 21-19
COUNt WAVeforms? 21-20
CTCJitter 20-14
CURSor? 19-3
DATA? 14-9
DATE 9-3
DCOLor 14-10
DEBug 9-4
DEFine 20-16
DELay 23-3
DELete 13-4, 21-21
DELTatime 20-20
DIFF 16-7
DIGitize 1-16, 8-7
DIRectory? 13-5
DISPlay 12-4, 16-8, 26-3
DIVide 16-9
DPRinter 17-4
DSP 9-6
DUTYcycle 20-22
ENABle 21-22
ERRor? 9-7
Event Status Enable 7-5
EXT PROBe 15-5
EXT PROBe ATTenuation 15-6
EXT PROBe EGAin 15-11
EXT PROBe EOFFset 15-12
EXT PROBe GAIN 15-13
EXT PROBe ID? 15-14
EXT PROBe SKEW 15-15
FACTors 17-6
FALLtime 20-24
FFT DFRequency 20-26
FFT FREQuency 16-10, 20-28
FFT MAGNitude 20-29
FFT PEAK1 20-30
FFT PEAK2 20-31
FFT RESolution 16-11
FFT THReshold 20-32
FFT WINDow 16-12
FFTMagnitude 16-14
FFTPhase 16-15
FREQuency 20-33
GPIB Mode 2-6
GRATicule 14-11
GRATicule INTensity 14-11
HAMPlitude 21-23
HEADer 9-8
HISTogram HITS 20-35
HISTogram M1S 20-41
HISTogram M2S 20-43
HISTogram M3S 20-45
HISTogram MEAN 20-37
HISTogram MEDian 20-39
HISTogram PEAK 20-47
HISTogram PP 20-49
HISTogram STDDev 20-51

HORizontal 16-16
HORizontal POSition 16-17
HORizontal RANGe 16-18
IMAGe 17-7
IMPedance 21-24
INPut 12-5, 15-4
INTegrate 16-19
INTerpolate 10-11
INVert 16-20, 21-26
JITTer:DIRection 20-53
JITTer:STATistics 20-55
LAMPlitude 21-27
LANGuage 9-12
LINE 14-13
LOAD 13-6, 21-28, 26-4
LONGform 9-13
MAGNify 16-21
MAXimum 16-22
MDIRectory 13-7
MEASure FFT DMAGnitude 20-27
MEASurement 16-23
MEASurement READout 19-4
MINimum 16-25
MODE 10-12, 18-5, 19-5
MODel? 8-10
MPRotect 11-8
MTEE 8-8
MULTiply 16-26
NREGions? 21-29
NWIDth 20-57
OFFSet 12-6, 16-27
OPEE 8-11
OPER? 8-12
Operation Complete (*OPC) 7-12
Option (*OPT) 7-13
OUTPut 11-9
OVERshoot 20-59
OVLenable 8-13
OVLRegister? 8-14
PERiod 20-61
PERSistence 14-14
PHASe 20-63
POINts 10-13
POINts AUTO 10-15
POSition 23-5
Power-on Status Clear (*PSC) 7-14
PREShoot 20-65
PRINt 8-15
PRINters? 17-8
PROBe 12-7, 15-5
PROBe ATTenuation 12-9
PROBe EGAin 12-14
PROBe EOFFset 12-15
PROBe GAIN 12-16
PROBe IMPedance? 21-30
PROBe PROTection CLEar 12-19
PROBe PROTection? 12-20
PROBe SKEW 12-18, 15-15
PWD? 13-8
PWIDth 20-67
RANGe 12-21, 15-16, 16-28, 23-6
Recall (*RCL) 7-15
RECall SETup 8-16
REFerence 23-7
Reset (*RST) 7-16
RESults? 20-69
RISetime 20-72
ROW 14-15
RUMode 21-31
SOFailure 21-33
RUN 8-17
SAVE 26-5
SCALe 12-22, 23-8
SCALe BIND 21-34
SCALe SIZE 18-6
SCALe X1 21-35
SCALe XDELta 21-36
SCALe Y1 21-37
SCALe Y2 21-38
SCOLor 14-16
SCOPETEST 22-5
SCRatch 20-74
SDONe? 11-10
SENDvalid 20-75
SERial 8-18
Service Request Enable (*SRE) 7-18
SETup 9-15
SIMage 13-9
SINGle 8-19
SKEW 11-11
SOURce 20-76, 21-39
SRATe 10-16
SRATe AUTO 10-18
SSAVer 14-19
SSAVer AAFTer 14-19
STARt 11-12
STARt | STOP 21-40
STATistics 20-77
STATus? 11-13
STIMe 21-41, 21-43
STOP 8-20
STORe 13-10
WAVEform 8-22
STORe SETup 8-21
STRing 14-20
SUBTract 16-29
TDELta? 19-6
TEDGe 20-78
TER? 8-23
TEXT 14-21
TIME 9-17
TITLe? 21-42
TMAX 20-80
TMIN 20-82
TRIG ADV COMM BWID 24-26
TRIG ADV COMM ENCode 24-27
TRIG ADV COMM LEVel 24-28
TRIG ADV COMM PATTern 24-29
TRIG ADV COMM POLarity 24-30
TRIG ADV COMM SOURce 24-31
TRIG ADV EDLY ARM SLOPe 24-46
TRIG ADV EDLY ARM SOURce 24-45
TRIG ADV EDLY EVENt DELay 24-47
TRIG ADV EDLY EVENt SLOPe 24-49
TRIG ADV EDLY EVENt SOURce 24-48
TRIG ADV EDLY TRIG SLOPe 24-51
TRIG ADV EDLY TRIG SOURce 24-50
TRIG ADV PATT CONDition 24-34
TRIG ADV PATT LOGic 24-35
TRIG ADV STATe CLOCk 24-38
TRIG ADV STATe CONDition 24-39
TRIG ADV STATe LOGic 24-40
TRIG ADV STATe LTYPe 24-41
TRIG ADV STATe SLOPe 24-42
TRIG ADV STV FIELd 24-61
TRIG ADV STV LINE 24-62
TRIG ADV STV SOURce 24-63
TRIG ADV STV SPOLarity 24-64
TRIG ADV TDLY ARM SLOPe 24-55

TRIG ADV TDLY ARM SOURce 24-54
TRIG ADV TDLY DELay 24-56
TRIG ADV TDLY TRIG SLOPe 24-58
TRIG ADV TDLY TRIG SOURce 24-57
TRIG ADV UDTV EDGE 24-68
TRIG ADV UDTV ENUMber 24-69
TRIG ADV UDTV PGTHan 24-70
TRIG ADV UDTV PLTHan 24-71
TRIG ADV UDTV POLarity 24-72
TRIG ADV UDTV SOURce 24-73
TRIG ADV VIOL MODE 24-76
TRIG ADV VIOL PWID DIR 24-81
TRIG ADV VIOL PWID POL 24-80
TRIG ADV VIOL PWID WIDT 24-82
TRIG ADV VIOL PWIDth 24-79
TRIG ADV VIOL SET HOLD DSO 24-97
TRIG ADV VIOL SET HOLD DSO HTHR 24-98
TRIG ADV VIOL SET HOLD DSO LTHR 24-99
TRIG ADV VIOL SET HOLD TIME 24-100
TRIG ADV VIOL SET MODE 24-86
TRIG ADV VIOL SET SET CSO 24-87
TRIG ADV VIOL SET SET CSO EDGE 24-89
TRIG ADV VIOL SET SET CSO LEV 24-88
TRIG ADV VIOL SET SET DSO 24-90
TRIG ADV VIOL SET SET DSO HTHR 24-91
TRIG ADV VIOL SET SET DSO LTHR 24-92
TRIG ADV VIOL SET SET TIME? 24-93
TRIG ADV VIOL SET SHOL CSO 24-101
TRIG ADV VIOL SET SHOL CSO EDGE 24-103
TRIG ADV VIOL SET SHOL CSO LEV 24-102
TRIG ADV VIOL SET SHOL DSO 24-104
TRIG ADV VIOL SET SHOL DSO HTHR 24-105
TRIG ADV VIOL SET SHOL DSO LTHR 24-106
TRIG ADV VIOL SET SHOL HTIMe 24-108
TRIG ADV VIOL SET SHOL STIMe 24-107
TRIG ADV VIOL TRAN 24-111
TRIG ADV VIOL TRAN SOUR 24-112
TRIG ADV VIOL TRAN SOUR HTHR 24-113
TRIG ADV VIOL TRAN SOUR LTHR 24-114
TRIG ADV VIOL TRAN TYPE 24-115
TRIG EDGE COUPling 24-17
TRIG EDGE SLOPe 24-18
TRIG EDGE SOURce 24-19
TRIG GLITch POLarity 24-22
TRIG GLITch SOURce 24-23
TRIG GLITch WIDTh 24-24
TRIG HOLDoff 24-9
TRIG HTHR 24-10
TRIG HYSTeresis 24-11
TRIG LEVel 24-12
TRIG LTHR 24-13
TRIG SWEep 24-14
Trigger (*TRG) 7-22
TRIGger EDGE SLOPe 24-15
TRIGger EDGE SOURce 24-15
TRIGger MODE 24-6, 24-8
TSTArt 19-7
TSTOp 19-9
TVOLt 20-83
UNITs 12-23, 15-17
VAMPlitude 20-85
VAVerage 20-87
VBASe 20-89
VDELta? 19-11
VERSus 16-30
VERTical 16-31
VIEW 8-24, 23-9
VIOL SET HOLD CSO 24-94
VIOL SET HOLD CSO EDGE 24-96
VIOL SET HOLD CSO LEV 24-95
VLOWer 20-91
VMAX 20-92
VMIDdle 20-94
VMIN 20-95
VPP 20-97
VRMS 20-99
VSTArt 19-12
VSTOp 19-14
VTIMe 20-101
VTOP 20-102
VUPPer 20-104
Wait-to-Continue (*WAI) 7-24
WAVeform BYTeorder 25-6
WAVeform DATA 25-11
WAVeform FORMat 25-32
WAVeform PREamble 25-36
WAVeform SOURce 25-41
WAVeform VIEW 25-44
WINDow DEFault 18-7
WINDow DELay 23-10
WINDow POSition 23-12
WINDow RANGe 23-13
WINDow SCALe 23-14
WINDow SOURce 18-8
WINDow X1Position|LLIMit 18-9
WINDow X2Position|RLIMit 18-10
WINDow Y1Position|TLIMit 18-11
WINDow Y2Position|BLIMit 18-12
X1Position 19-16
X1Y1source 19-18
X2Position 19-17
X2Y2source 19-19
XOFFset 26-6
XRANge 26-7
Y1Position 19-21
Y2Position 19-22
YOFFset 26-8
YRANge 26-9
command
execution and order 3-4
structure 1-15
Command and Data Concepts
GPIB 2-6
Command Error 29-5
Status Bit 4-3
command language
HP 545XX Oscilloscopes 28-2
HP 547XX Oscilloscopes 27-2
Command Tree 5-6, 5-8, 5-9, 5-11
Command Types 5-6
commands
SYSTem LANGuage 27-2, 28-2
commands embedded in program

messages 1-13
commas and spaces 1-5
comma-separated
variable file format 6-15
Common Command Header 1-7
Common Commands 7-2
Clear Status (*CLS) 7-4
Event Status Enable (*ESE) 7-5
Event Status Register (*ESR) 7-7
Identification Number (*IDN) 7-9
Learn (*LRN) 7-10
Operation Complete (*OPC) 7-12
Option (*OPT?) 7-13
Power-on Status Clear (*PSC?) 7-14
Recall (*RCL) 7-15
Reset (*RST) 7-16
Save (*SAV) 7-17
Service Request Enable (*SRE) 7-18
Status Byte (*STB?) 7-20
Test (*TST?) 7-23
Trigger (*TRG) 7-22
Wait-to-Continue (*WAI) 7-24
within a program message 7-3
Communicating Over the GPIB Interface 2-7
Communicating Over the LAN Interface 2-8
COMPlete 10-7
COMPlete query 25-8
COMPlete STATe 10-9
compound command header 1-6
compound queries 3-4
Computer Code and Capability 2-5
concurrent commands 5-13
CONDition
and STATe 24-39
in TRIG ADV PATTern 24-34
in TRIG ADV STATe 24-39
CONFig 10-10
CONNect 14-8
CONTinue 11-7
conventions of programming 5-2
converting waveform data
from data value to Y-axis units 25-4
sample program 6-14
COUNt 10-5
in MTESt AVERage command 21-17
COUNt query 25-9
COUPling
in TRIGger EDGE 24-17
COUPling query 25-10
coupling, input 12-5, 15-4
CREate
in MTESt AMASk command 21-7
CROSsing
in MEASure CGRade command 20-7
CTCJitter
in MEASure command 20-14
CURSor? 19-3

**D**

DATA 25-11
data
acquisition 25-3
conversion 25-4
flow 5-3
data in a learnstring 1-4
data in a program 1-5
Data Mode
GPIB 2-6
Data Structures
and Status Reporting 4-5
data transmission mode
and FORMat 25-32
DATA? 14-9
DATE 9-3
DCDistortion
in MEASure CGRade command 20-8
DCOLor 14-10
DDE bit 7-6, 7-8
DEBug 9-4
decimal 32 (ASCII space) 1-5
Decision Chart for Status Reporting 4-19
DEFault
in HISTogram WINDow command 18-7
Default
GPIB Address 2-7
Startup Conditions 2-4
Default Startup Conditions 2-4
DEFine 20-16
defining functions 16-2
def-length block response data 1-20
DELay 23-3
in TRIG ADV EDLY EVENt 24-47
in TRIG ADV TDLY 24-56
delay
and WINDow DELay 23-10
delay trigger modes 24-43, 24-52
DELete 13-4, 21-21
deleting files 13-4
DELTatime 20-20
and DEFine 20-16
derivative of functions 16-7
Device Address
GPIB 2-7
LAN 2-8
device address 1-3, 1-4
Device Clear (DCL) 2-9
Device Clear Code and Capability 2-5
Device Dependent Error (DDE), Status Bit 4-4
Device- or Oscilloscope-Specific Error 29-7
Device Trigger Code and Capability 2-5
device-dependent data 1-20
DFREQuency
in MEASure FFT command 20-26
DIFF 16-7
digital bandwidth limit filter 10-6
DIGitize 8-7
setting up for execution 10-2
Digitize
Aborting 2-9
DIRectory? 13-5
Disabling Serial Poll 2-9
discrete derivative of functions 16-7
Disk Commands 13-2
CDIRectory 13-3
DELete 13-4
DIRectory? 13-5
LOAD 13-6
MDIRectory 13-7
PWD? 13-8
SIMage 13-9
STORe 13-10
DISPlay 12-4, 16-8, 26-3
DISPlay Commands
CGRade 14-3
CGRADE LEVels? 14-5
CGRade LEVels? 14-5
Display Commands 14-2
COLumn 14-7
CONNect 14-8

DATA? 14-9
DCOLor 14-10
GRATicule 14-11
GRATicule INTensity 14-11
LINE 14-13
PERSistence 14-14
ROW 14-15
SCOLor 14-16
SSAVer 14-19
SSAVer AAFTer 14-19
STRing 14-20
TEXT 14-21
display persistence 14-14
DIVide 16-9
dividing functions 16-9
DMAGnitude
in MEASure FFT command 20-27
DPRinter 17-4
Driver Electronics Code and Capability 2-5
DSP (display) 9-6
duplicate mnemonics 1-8
DUTYcycle 20-22

**E**

EADapter 12-10, 15-7
ECoupling 12-12, 15-9
EDGE
in TRIG ADV UDTV 24-68
trigger mode 24-15
EDGE trigger commands 24-15
EHEight
in MEASure CGRade command 20-9
Ellipsis
... 1-5
embedded
commands 1-13
strings 1-3, 1-4, 1-12
ENABle 21-22
Enable Register 7-3
ENCode
in TRIG ADV COMM 24-27
End Of String (EOS) 1-12
End Of Text (EOT) 1-12
End-Or-Identify (EOI) 1-12
ENUMber
in TRIG ADV UDTV 24-69
EOI and IEEE 488.2 5-13
equipment for calibration 11-3
equivalent time mode 10-12
error
in measurements 20-4
messages 29-2
numbers 29-4
query interrupt 1-9, 1-18
error checking
sample program 6-10
Error Messages table 29-9
error queue 29-3
and status reporting 4-17
overflow 29-3
ERRor? 9-7
errors
exceptions to protocol 3-4
ESB (Event Status Bit) 4-4, 7-19, 7-21
ESB (Event Summary Bit) 7-5
*ESE (Event Status Enable) 7-5
ESR (Standard Event Status Register) 4-11
ETIMe 10-12
event monitoring 4-2
Event Registers Default 2-4
Event Status Bit (ESB) 4-4
Event Status Enable (*ESE)
Status Reporting 4-12
Event Summary Bit (ESB) 7-5
EWIDth
in MEASure CGRade command 20-10
EWINdow, and DEFine 20-17
Example Program 1-15
in initialization 1-15
example programs
C and BASIC 6-2
exceptions to protocol 3-4
EXE bit 7-6, 7-8
executing DIGITIZE 10-2
execution
errors, and command errors 29-5
of commands and order 3-4
Execution Error 29-6
Execution Error (EXE), Status Bit 4-3
exponential notation 1-11
exponents 1-11
External Channel Commands 15-2
BWLimit 15-3
INPut 15-4
PROBe 15-5
PROBe SKEW 15-15
RANGe 15-16
UNITs 15-17
External Commands
EADapter 15-7
PROBe EGAin 15-11
PROBe EOFFset 15-12
PROBe ID? 15-14
EXTernal PROBe ID? 15-14

**F**

FACTors 17-6
FAILures?
in MTESt COUNt command 21-18
fall time measurement setup 20-4
FALLtime 20-24
FFT Commands 20-4
FFTMagnitude 16-14
FFTPhase 16-15
FIELd
in TRIG ADV STV 24-61
filter, internal low-pass 12-3, 15-3
filtering 10-6
flow of acquired data 5-3
FORMat 25-32
and DATA 25-13
formatting query responses 9-2
fractional values 1-11
FREQuency 20-33
in FUNCtion FFT command 16-10
in MEASure FFT command 20-28
frequency measurement setup 20-4
full-scale vertical axis 12-21
FUNCtion 16-4
function
and vertical scaling 16-28
time scale 16-3
Function Commands 16-2
ADD 16-5
AVERage 16-6
DIFF 16-7
DISPlay 16-8
DIVide 16-9
FFT FREQuency 16-10
FFT RESolution 16-11
FFT WINDow 16-12

FFTMagnitude 16-14
FFTPhase 16-15
FUNCtion? 16-4
HORizontal 16-16
HORizontal POSition 16-17
HORizontal RANGe 16-18
INTegrate 16-19
INVert 16-20
MAGNify 16-21
MAXimum 16-22
MEASurement 16-23
MINimum 16-25
MULTiply 16-26
OFFSet 16-27
RANGe 16-28
SUBTract 16-29
VERSus 16-30
VERTical 16-31
functional elements of protocol 3-3
functions
and VIEW 25-44
combining in instructions 1-7
FWAVeforms?
in MTESt COUNt command 21-19

**G**

gain and offset of a probe 11-4
generating service request
sample program 6-16, 6-18, 6-19
GLITch
trigger mode 24-21
glitch
trigger mode 24-20
GPIB
Interface Connector 2-3
GRATicule 14-11
HARDcopy AREA 17-3
Group Execute Trigger (GET) 2-9

**H**

Halting bus activity 2-9
HAMPlitude 21-23
Hardcopy Commands 17-2
AREA 17-3
DPRinter 17-4
FACTors 17-6
IMAGe 17-7
PRINters? 17-8
hardcopy of the screen 17-2
hardcopy output and message
termination 3-4
HEADer 9-8
header
stripped 6-13
within instruction 1-4
headers 1-4
types 1-6
HELP HEADers 9-10
Histogram Commands 18-2
AXIS 18-4
MODE 18-5
SCALe SIZE 18-6
WINDow DEFault 18-7
WINDow SOURce 18-8
WINDow X1Position|LLIMit 18-9
WINDow X2Position|RLIMit 18-10
WINDow Y1Position|TLIMit 18-11
WINDow Y2Position|BLIMit 18-12
HITS
in MEASure HISTogram command 20-35
HOLDoff
in TRIGger 24-9
HORizontal 16-16
horizontal
functions, controlling 23-2
offset, and XOFFset 26-6
range, and XRANge 26-7
scaling and functions 16-3
HORizontal POSition 16-17
HORizontal RANGe 16-18
Host language 1-4
HP 545XX Commands
Acquire 28-5
Calibrate 28-6
Channel 28-7
Common 28-31
Disk 28-8
Display 28-9
External 28-10
FFT (n/a) 28-11
Function 28-12
Hardcopy 28-13
Limit Test (n/a) 28-14
Marker 28-15
Measure 28-16
Memory Test (n/a) 28-19
Multiple Memory (n/a) 28-18
Pixel Memory (n/a) 28-20
Root 28-30
Self-Test 28-21
Sequential (n/a) 28-22
System 28-23
Time Base 28-24
Trigger 28-25
Waveform 28-28
Waveform Memory 28-29
HP 547XX Commands
Acquire 27-5
Calibrate 27-6
Channel 27-7
Common 27-30
Disk 27-8
Display 27-9
External 27-10
FFT 27-11
Function 27-12
Hardcopy 27-13
Limit Test (n/a) 27-14
Marker 27-15
Measure 27-16
Memory Test (n/a) 27-18
Multiple Memory (n/a) 27-17
Pixel Memory (n/a) 27-19
Root 27-29
Self-Test 27-20
Sequential (n/a) 27-21
System 27-22
Time Base 27-23
Trigger 27-24
Waveform 27-27
Waveform Memory 27-28
HP BASIC 5.0 1-2
HTHReshold 24-10
hue 14-17
HYSTeresis
in TRIGger 24-11

**I**

*IDN? (Identification Number) 7-9
IEEE 488.1 3-2
and IEEE 488.2 relationship 3-2
IEEE 488.2 3-2
compliance 3-2

conformity 1-2
Standard 1-2
Standard Status Data Structure Model 4-2
IMAGe 17-7
image specifier, -K 9-16
image specifiers
and DATA 25-11
and PREamble 25-37
IMPedance 21-24
impedance, input 12-5, 15-4
IMPedance?
in MTESt PROBe command 21-30
individual commands language 1-2
Infinity Representation 5-13
initialization 1-14
event status 4-2
IO routine 6-5
sample program 6-4
initializing oscilloscope
sample program 6-6, 6-17
INPut 12-5, 15-4
Input Buffer
Clearing 2-9
input buffer 3-3
default condition 3-4
input coupling
and COUPling? 25-10
instruction headers 1-4
Instrument Address
GPIB 2-7
instrument status 1-21
integer definition 1-11
INTegrate 16-19
intensity 14-11
Interface
Capabilities 2-5
Clear (IFC) 2-9
GPIB Select Code 2-7
interface
functions 2-2
interface, initializing 1-14
internal low-pass filter 12-3, 15-3
INTerpolate 10-11
interpreting commands, parser 3-3
interrupted query 1-9, 1-18
Introduction to Programming 1-2
INVert 16-20, 21-26
inverting functions 16-20

**J**

JITTer
in MEASure CGRade command 20-11
JITTer:DIRection 20-53
JITTer:STATistics 20-55

**K**

-K 9-16
K, and DATA 25-11

**L**

LAMPlitude 21-27
LANGuage 9-12
language
HP 545XX Oscilloscopes 28-2
HP 547XX Oscilloscopes 27-2
language for program examples 1-2
Learn (*LRN) 7-10
learnstring block data 1-4
LEVel
in TRIG ADV COMM 24-28
in TRIGger 24-12
LEVels?
in DISPlay CGRade command 14-5
LF/HF reject, input 12-5, 15-4
LINE 14-13
in TRIG ADV STV 24-62
linefeed 1-12
List of Error Messages 29-9
Listener Code and Capability 2-5
Listeners, Unaddressing All 2-9
LOAD 13-6, 21-28, 26-4
loading and saving 13-2
LOGic
and STATe 24-40
in TRIG ADV PATT 24-35
in TRIG ADV STATe 24-40
LONG
and FORMat 25-33
LONGform 9-13
long-form headers 1-10
lowercase 1-10
headers 1-10
low-pass filter, internal 12-3, 15-3
*LRN (Learn) 7-10
*LRN?
and SYSTem SETup? 9-16
LSBFirst, and BYTeorder 25-6
LTHReshold 24-13
LTYPe
and STATe 24-41
in TRIG ADV STATe 24-41
luminosity 14-17

**M**

M1S
in MEASure HISTogram command 20-41
M2S
in MEASure HISTogram command 20-43
M3S
in MEASure HISTogram command 20-45
MAGNify 16-21
MAGNitude
in MEASure FFT command 20-29
MAIN, and VIEW 25-44
making measurements 20-4
Marker Commands 19-2
CURSor? 19-3
MEASurement READout 19-4
MODE 19-5
TDELta? 19-6
TSTArt 19-7
TSTOp 19-9
VDELta? 19-11
VSTArt 19-12
VSTOp 19-14
X1Position 19-16
X1Y1source 19-18
X2Position 19-17
X2Y2source 19-19
XDELta? 19-20
Y1Position 19-21
Y2Position 19-22
YDELta? 19-23
Mask Test Commands 21-2
ALIGn 21-4
AlignFIT 21-5
AMASk CREate 21-7
AMASk SAVE|STORe 21-9
AMASk SOURce 21-8

AMASk UNITs 21-10
AMASk XDELta 21-11
AMASk YDELta 21-13
AUTO 21-15
AVERage 21-16
AVERage COUNt 21-17
COUNt FAILures? 21-18
COUNt FWAVeforms? 21-19
COUNt WAVeforms? 21-20
DELete 21-21
ENABle 21-22
HAMPlitude 21-23
IMPedance 21-24
INVert 21-26
LAMPlitude 21-27
LOAD 21-28
NREGions? 21-29
PROBe IMPedance? 21-30
RUMode 21-31
RUMode SOFailure 21-33
SCALe
BIND 21-34
Y1 21-37
SCALe X1 21-35
SCALe XDELta 21-36
SCALe Y1 21-37
SCALe Y2 21-38
SOURce 21-39
STARt | STOP 21-40
STIMe 21-41, 21-43
TITLe? 21-42
mask, Service Request Enable Register 7-18
Master Summary Status (MSS)
and *STB 7-20
Status Bit 4-4
MAV (Message Available) 4-4
bit 7-19, 7-21
MAXimum 16-22
MDIRectory 13-7
MEAN
in MEASure HISTogram command 20-37
MEASure
RESults and statistics 20-77
Measure Commands 20-2
AREA 20-6
CGRade CROSsing 20-7
CGRade DCDistortion 20-8
CGRade EHEight 20-9
CGRade EWIDth 20-10
CGRade JITTer 20-11
CGRade QFACtor 20-12
CLEar 20-13
CTCJitter 20-14
DEFine 20-16
DELTatime 20-20
DUTYcycle 20-22
FALLtime 20-24
FFT DFRequency 20-26
FFT DMAGnitude 20-27
FFT FREQuency 20-28
FFT MAGNitude 20-29
FFT PEAK1 20-30
FFT PEAK2 20-31
FFT THReshold 20-32
FREQuency 20-33
HISTogram HITS 20-35
HISTogram M1S 20-41
HISTogram M2S 20-43
HISTogram M3S 20-45
HISTogram MEAN 20-37
HISTogram MEDian 20-39
HISTogram PEAK 20-47
HISTogram PP 20-49
HISTogram STDDev 20-51
JITTer:DIRection 20-53
JITTer:STATistics 20-55
NWIDth 20-57
OVERshoot 20-59
PERiod 20-61
PHASe 20-63
PREShoot 20-65
PWIDth 20-67
RESults? 20-69
RISetime 20-72
SCRatch 20-74
SENDvalid 20-75
SOURce 20-76
STATistics 20-77
TEDGe 20-78
TMAX 20-81
TMIN 20-82
TVOLt 20-83
VAMPlitude 20-85
VAVerage 20-87
VBASe 20-89
VLOWer 20-91
VMAX 20-92
VMIDdle 20-94
VMIN 20-95
VPP 20-97
VRMS 20-99
VTIMe 20-101
VTOP 20-102
VUPPer 20-104
MEASurement 16-23
measurement
error 20-4
readout 19-4
setup 20-4
source 20-76
MEDian
in MEASure HISTogram command 20-39
memories, and VIEW 25-44
message
queue 4-18
termination with hardcopy 3-4
Message (MSG), Status Bit 4-4
Message Available (MAV)
and *OPC 7-12
Status Bit 4-4
Message Communications and System Functions 3-2
Message Event Register 4-10
message exchange protocols
of IEEE 488.2 3-3
MIN 16-25
Mnemonic Truncation 5-5
MODE 10-12, 18-5, 19-5
in TRIGger MODE 24-6, 24-8
MODel? 8-10
monitoring events 4-2
MPRotect 11-8
MSBFirst, and BYTeorder 25-6
MSG
bit in the status register 4-10
MSG bit 7-19, 7-21
MSS bit and *STB 7-20
MTEE 8-8
multiple
program commands 1-13
queries 1-21

subsystems 1-13
Multiple numeric variables 1-21
MULTiply 16-26

N
NL (New Line) 1-12
NREGions? 21-29
NTSC TV trigger mode 24-59
numeric
program data 1-11
variable example 1-19
variables 1-19
NWIDth 20-57

O
OFFSet 12-6, 16-27
offset and gain of a probe 11-4
*OPC (Operation Complete) 7-12
OPC bit 7-6, 7-8
OPEE 8-11
OPER bit 7-19, 7-21
OPER query 8-12
operands and time scale 16-3
operating the disk 13-2
Operation Complete (*OPC) 7-12
Status Bit 4-4
operation status 4-2
*OPT (Option) 7-13
Options, Program Headers 1-10
order of commands and execution 3-4
oscilloscope
trigger modes and commands 24-6
Oscilloscope Default GPIB Address 2-7
OUTPut 11-9
output buffer 1-9, 1-18
Output Command 1-4
Output Queue
Clearing 2-9
output queue 1-9, 4-17
default condition 3-4
definition 3-3
OUTPUT statement 1-3
overlapped and sequential commands 5-13
OVERshoot 20-59
OVLenable 8-13
OVLRegister query 8-14

P
PAL-M TV trigger mode 24-59
Parallel Poll Code and Capability 2-5
parametric measurements 20-2
Parser
Resetting 2-9
parser 1-14, 3-3
default condition 3-4
definition 3-3
passing values across the bus 1-9
passive probes and calibration 11-4
PATTern
in TRIG ADV COMM 24-29
PDETect 10-12
PEAK
in MEASure HISTogram command 20-47
PEAK1
in MEASure FFT command 20-30
PEAK2
in MEASure FFT command 20-31
peak-to-peak voltage, and VPP 20-97
Pending Commands, Clearing 2-9
PERiod 20-61
period measurement setup 20-4
PERsistence 14-14
PGTHan
in TRIG ADV UDTV 24-70
PHASe 20-63
PLTHan
in TRIG ADV UDTV 24-71
POINts 10-13
POINts AUTO 10-15
POINts query 25-35
POLarity
and GLITch 24-22
in TRIG ADV COMM 24-30
in TRIG ADV UDTV 24-72
in TRIGger GLITch 24-22
PON bit 7-8
POSition 23-5
position
and WINDow POSition 23-12
pound sign (#) and block data 1-20
Power On (PON) status bit 4-3, 7-6
Power-up Condition 2-4
PP
in MEASure HISTogram command 20-49
PREamble 25-36
and DATA 25-13
PREShoot 20-65
PRINt 8-15
PRINters? 17-8
printing
specific screen data 17-3
the screen 17-2
PROBe 12-7, 15-5
PROBe ATTenuation 12-9, 15-6
probe attenuation factor 11-4
Probe Calibration 11-4
PROBe EGAin 12-14, 15-11
PROBe EOFFset 12-15, 15-12
PROBe GAIN 12-16, 15-13
PROBe PROTection CLEar 12-19
PROBe PROTection? 12-20
PROBe SKEW 12-18, 15-15
program data 1-5
Program example 1-15
Program Header Options 1-10
program message terminator 1-12
program overview
initialization example 1-15
programming basics 1-2
Programming Conventions 5-2
programming examples language 1-2
Programming Getting Started 1-13
protocol
exceptions and operation 3-4
*PSC (Power-on Status Clear) 7-14
pulse width measurement setup 20-4
pulse width violation mode 24-77
PWD? 13-8
PWIDth 20-67

Q
QFACtor
in MEASure CGRade command 20-12
Query
*SRE? 7-18
quantization levels 6-14
Query 1-4, 1-9
*ESE? (Event Status Enable) 7-5
*ESR? (Event Status Register) 7-7
*STB? (Status Byte) 7-20

AER? 8-3
AREA? 17-3
AttenSET? 22-3
AVERage? 10-4
BANDpass? 25-5
BWLimit? 10-6, 12-3, 15-3
BYTeorder? 25-6
CHANnel PROBe ID? 12-17
CLIPped? 25-7
COLumn? 14-7
COMPlete STATe? 10-9
COMPlete? 10-8, 25-8
CONFig? 10-10
CONNect? 14-8
COUNt? 10-5, 21-17, 25-9
COUPling? 25-10
CURSor? 19-3
DATA? 14-9, 25-12
DATE? 9-3
DEBug? 9-5
DELay? 23-4
DELTatime? 20-21
DIRectory? 13-5
DISPlay? 12-4, 16-8, 26-3
DPRinter? 17-5
DSP? 9-6
DUTYcycle? 20-23
EADapter? 12-11
ECoupling? 12-13, 15-10
ERRor? 9-7
EXT INPut? 15-4
EXT PROBe ATTenuation? 15-6
EXT PROBe EADapter? 15-8
EXT PROBe EGAin? 15-11
EXT PROBe EOFFset? 15-12
EXT PROBe ID? 15-14
EXT PROBe SKEW? 15-15
EXT PROBe? 15-5
EXT RANGe? 15-16
EXT UNITs? 15-17
FACTors? 17-6
FALLtime? 20-25
FFT RESolution? 16-11
FORMat? 25-34
FREQuency? 20-34
FUNCtion? 16-4
GRATicule? 14-12
HEADer 9-8
HELP HEADers? 9-10
HORizontal POSition? 16-17
HORizontal RANGe? 16-18
HORizontal? 16-16
Identification Number (*IDN?) 7-9
IMAGe? 17-7
INPut? 12-5
INTerpolate? 10-11
JITTer:DIRection? 20-53
JITTer:STATistics? 20-56
LANGuage? 9-12
Learn (*LRN?) 7-10
LONGform? 9-13
MEASure FALLtime? 20-25
MEASure FFT DFRequency? 20-26
MEASure FFT DMAGnitude? 20-27
MEASure FFT FREQuency? 20-28
MEASure FFT MAGNitude? 20-29
MEASure FFT PEAK1? 20-30
MEASure FFT PEAK2? 20-31
MEASure FFT THReshold? 20-32
MEASurement? 16-24
MODE? 10-12, 19-5
MODel? 8-10
MPRotect? 11-8
NWIDth? 20-58
OFFSet? 12-6, 16-27
Option (*OPT?) 7-13
OUTPut? 11-9
OVERshoot? 20-60
PERiod? 20-62
PERSistence? 14-14
PHASe? 20-64
POINts AUTO? 10-15
POINts? 10-14, 25-35
POSition? 23-5
Power-on Status Clear (*PSC?) 7-14
PREamble? 25-38
PREShoot? 20-66
PRINters? 17-8
PROBe ATTenuation? 12-9
PROBe EGAin? 12-14
PROBe EOFFset? 12-15
PROBe GAIN? 12-16, 15-13
PROBe SKEW? 12-18
PROBe? 12-8
PWD? 13-8
PWIDth? 20-68
RANGe? 12-21, 16-28, 23-6
REFerence? 23-7
RESults? 20-69
RISetime? 20-73
ROW? 14-15
SCALe? 12-22, 23-8
SCOLor? 14-18
SCOPETEST? 22-5
SDONe? 11-10
SENDvalid? 20-75
SETup? 9-15
SKEW? 11-11
SOURce? 20-76, 25-41
SRATe AUTO? 10-18
SRATe? 10-17
SSAVer AAFTer? 14-19
SSAVer? 14-19
STATistics? 20-77
Status Byte (*STB) 7-20
STATus? 11-13
TDELta? 19-6
TEDGe? 20-79
TER? 8-23
Test (*TST?) 7-23
TMAX? 20-81
TMIN? 20-82
TRIG ADV COMM BWID? 24-26
TRIG ADV COMM ENCode? 24-27
TRIG ADV COMM LEVel? 24-28
TRIG ADV COMM PATTern? 24-29
TRIG ADV COMM POLarity? 24-30
TRIG ADV COMM SOURce? 24-31
TRIG ADV EDLY ARM SLOPe? 24-46
TRIG ADV EDLY ARM SOURce 24-45
TRIG ADV EDLY EVENt DELay? 24-47
TRIG ADV EDLY EVENt SLOPe? 24-49
TRIG ADV EDLY EVENt SOURce? 24-48
TRIG ADV EDLY TRIG SLOPe? 24-51
TRIG ADV EDLY TRIG SOURce? 24-50
TRIG ADV PATT COND? 24-34
TRIG ADV PATT LOGic? 24-35
TRIG ADV STATe CLOCk? 24-38

TRIG ADV STATe CONDition? 24-39
TRIG ADV STATe LOGic? 24-40
TRIG ADV STATe LTYPe? 24-41
TRIG ADV STATe SLOPe? 24-42
TRIG ADV STV FIELd? 24-61
TRIG ADV STV LINE? 24-62
TRIG ADV STV SOURce? 24-63
TRIG ADV STV SPOLarity? 24-64
TRIG ADV TDLY ARM SLOPe? 24-55
TRIG ADV TDLY ARM SOURce? 24-54
TRIG ADV TDLY DELay? 24-56
TRIG ADV TDLY TRIG SLOPe? 24-58
TRIG ADV TDLY TRIG SOURce? 24-57
TRIG ADV UDTV EDGE? 24-68
TRIG ADV UDTV ENUMber? 24-69
TRIG ADV UDTV PGTHan? 24-70
TRIG ADV UDTV PLTHan? 24-71
TRIG ADV UDTV POLarity? 24-72
TRIG ADV UDTV SOURce? 24-74
TRIG ADV VIOL MODE? 24-76
TRIG ADV VIOL PWID DIR? 24-81
TRIG ADV VIOL PWID POL? 24-80
TRIG ADV VIOL PWID WIDT? 24-82
TRIG ADV VIOL PWIDth? 24-79
TRIG ADV VIOL SET HOLD CSO EDGE? 24-96
TRIG ADV VIOL SET HOLD CSO LEV? 24-95
TRIG ADV VIOL SET HOLD CSO? 24-94
TRIG ADV VIOL SET HOLD DSO HTHR? 24-98
TRIG ADV VIOL SET HOLD DSO LTHR? 24-99
TRIG ADV VIOL SET HOLD DSO? 24-97
TRIG ADV VIOL SET HOLD TIME? 24-100
TRIG ADV VIOL SET MODE? 24-86
TRIG ADV VIOL SET SET CSO EDGE? 24-89
TRIG ADV VIOL SET SET CSO LEV? 24-88
TRIG ADV VIOL SET SET CSO? 24-87
TRIG ADV VIOL SET SET DSO HTHR? 24-91
TRIG ADV VIOL SET SET DSO LTHR? 24-92
TRIG ADV VIOL SET SET DSO? 24-90
TRIG ADV VIOL SET SET TIME? 24-93
TRIG ADV VIOL SET SHOL CSO EDGE? 24-103
TRIG ADV VIOL SET SHOL CSO LEV? 24-102
TRIG ADV VIOL SET SHOL CSO? 24-101
TRIG ADV VIOL SET SHOL DSO HTHR? 24-105
TRIG ADV VIOL SET SHOL DSO LTHR? 24-106
TRIG ADV VIOL SET SHOL DSO? 24-104
TRIG ADV VIOL SET SHOL HTIMe? 24-108
TRIG ADV VIOL SET SHOL STIMe? 24-107
TRIG ADV VIOL TRAN SOUR HTHR? 24-113
TRIG ADV VIOL TRAN SOUR LTHR? 24-114
TRIG ADV VIOL TRAN SOUR? 24-112
TRIG ADV VIOL TRAN TYPE? 24-115
TRIG ADV VIOL TRAN? 24-111
TRIG EDGE COUPling? 24-17
TRIG EDGE SLOPe? 24-18
TRIG EDGE SOURce? 24-19
TRIG GLITch POLarity? 24-22
TRIG GLITch SOURce? 24-23
TRIG HOLDoff? 24-9
TRIG HTHR? 24-10
TRIG HYSTeresis? 24-11
TRIG LEVel? 24-12
TRIG LTHR? 24-13
TRIG SWEep? 24-14
TRIGger GLITch WIDTh? 24-24
TRIGger MODE? 24-8
TSTArt? 19-7
TSTOp? 19-10
TVOLt? 20-83
TYPE? 25-42
UNITs? 12-23
VAMPlitude? 20-86
VAVerage? 20-88
VBASe? 20-90
VDELta? 19-11
VIEW? 23-9, 25-45
VLOWer? 20-91
VMAX? 20-93
VMIDdle? 20-94
VMIN? 20-96
VPP? 20-98
VRMS? 20-100
VSTArt? 19-12
VSTOp? 19-14
VTIMe? 20-101
VTOP? 20-103
VUPPer? 20-105
WINDow DELay? 23-11
WINDow POSition? 23-12
WINDow RANGe? 23-13
WINDow SCALe? 23-14
X1Position? 19-16
X1Y1source? 19-18
X2Position? 19-17
X2Y2source? 19-19
XDELta? 19-20
XDISplay? 25-46
XINCrement? 25-47
XOFFset? 26-6
XORigin? 25-48
XRANge? 25-49, 26-7
XREFerence? 25-50
XUNits? 25-51
Y1Position? 19-21
YDELta? 19-23
YDISplay? 25-52
YINCrement? 25-53
YOFFset? 26-8
YORigin? 25-54
YRANge? 25-55, 26-9
YREFerence? 25-56
YUNits? 25-57
query
  headers 1-9
  interrupt 1-9
  response 1-18

responses, formatting 9-2
Query Error 29-8
QYE Status Bit 4-4
query interrupt 1-18
question mark 1-9
queue, output 1-9
quoted strings 14-13
quotes, with embedded strings 1-12
QYE bit 7-6, 7-8

**R**

RANGe 12-21, 15-16, 16-28, 23-6
range
and WINDow RANGe 23-13
*RCL (Recall) 7-15
README file
for sample programs 6-20
real number definition 1-11
real time mode 10-12
and interpolation 10-11
RECall 8-16
Receiving Common Commands 7-3
Receiving Information from the Instrument 1-18
REFerence 23-7
register
save/recall 7-15, 7-17
Standard Event Status Enable 4-12
reliability of measured data 4-2
Remote Local Code and Capability 2-5
remote programming basics 1-2
REPetitive 10-12
representation of infinity 5-13
Request Control (RQC)
Status Bit 4-4
Request Service (RQS)
Default 2-4
status bit 4-4
Reset (*RST) 7-16
Resetting the Parser 2-9
RESolution
in FUNCtion FFT command 16-11
response
data 1-20
generation 5-13
responses, buffered 5-13
result state code, and SENDvalid 20-75
RESults? 20-69
Returning control to system computer 2-9
rise time measurement setup 20-4
RISetime 20-72
RMS voltage, and VRMS 20-99
Root level commands 8-2
AER? 8-3
AUToscale 8-4
BLANk 8-5
CDISplay 8-6
DIGitize 8-7
MODel? 8-10
MTEE 8-8
OPEE 8-11
OPER? 8-12
OVLenable 8-13
OVLEnable? 8-13
OVLRegister? 8-14
PRINt 8-15
RECall 8-16
RUN 8-17
SERial 8-18
SINGle 8-19
STOP 8-20
STORe 8-21
STORe WAVEform 8-22
TER? 8-23
VIEW 8-24
ROW 14-15
RQC (Request Control) 4-4
bit 7-6, 7-8
RQS (Request Service) 4-4
and *STB 7-20
Default 2-4
RQS/MSS bit 7-21
*RST (Reset) 6-17, 7-16
RTIMe 10-12
rule of truncation 5-5
rules of traversal 5-7
RUMode 21-31
RUN 8-17
and GET relationship 2-9

**S**

sample programs 6-2
segments 6-3
sample rate 10-16
and bandwidth limit 10-6
sampling mode 10-12
saturation 14-17
*SAV (Save) 7-17
SAVE 26-5
save/recall register 7-15, 7-17
SAVE|STORe
in MTESt AMASk command 21-9
saving and loading 13-2
SCALe 12-22, 23-8
Y1 21-37
SCOLor 14-16
SCOPETEST
in self-test commands 22-5
SCRatch 20-74
SCReen
HARDcopy AREA 17-3
SDONe? 11-10
segments of sample programs 6-3
Selected Device Clear (SDC) 2-9
Selecting Multiple Subsystems 1-13
self test 7-23
Self-Test Commands 22-2
AttenSET? 22-3
CANCel 22-4
SCOPETEST 22-5
semicolon usage 1-7
sending compound queries 3-4
SENDvalid 20-75
separator 1-5
Sequential and Overlapped Commands 5-13
SERial (SERial number) 8-18
Serial Poll
Disabling 2-9
serial poll
(SPOLL) in example 4-9
of the Status Byte Register 4-9
serial prefix, reading 7-9
Service Request
Code and Capability 2-5
sample program 6-16
Service Request Enable
(*SRE) 7-18
Register (SRE) 4-10
Register Bits 7-19
Register Default 2-4
setting
bits in the Service Request Enable

Register 4-10
horizontal tracking 16-16
Standard Event Status Enable Register bits 4-12
time and date 9-17
TRG bit 4-10
voltage and time markers 19-2
setting up
for programming 1-13
service request 6-18
the instrument 1-14
SETup 9-15
setup recall 7-15
setup violation mode 24-83
setup, storing 13-10
Short form 1-10
short-form headers 1-10
short-form mnemonics 5-5
SIMage 13-9
simple command header 1-6
SINGle 8-19
SIZE
in HISTogram SCALe command 18-6
SKEW, in CALibrate command 11-11
SLOPe
and STATe 24-42
in TRIG ADV EDLY ARM 24-46
in TRIG ADV EDLY EVENt 24-49
in TRIG ADV EDLY TRIGger 24-51
in TRIG ADV STATe 24-42
in TRIG ADV TDLY ARM 24-55
in TRIG ADV TDLY TRIGger 24-58
in TRIGger EDGE 24-18
SOFailure
in MTESt RUMode command 21-33
software version, reading 7-9
SOURce 20-76, 21-39, 25-41
and GLITch 24-23
and measurements 20-5
in HISTogram WINDow command 18-8
in MTEST AMASk command 21-8
in TRIG ADV COMM 24-31
in TRIG ADV EDLY ARM 24-45
in TRIG ADV EDLY EVENt 24-48
in TRIG ADV EDLY TRIGger 24-50
in TRIG ADV STV 24-63
in TRIG ADV TDLY ARM 24-54
in TRIG ADV TDLY TRIGger 24-57
in TRIG ADV UDTV 24-73
in TRIGger EDGE 24-19
in TRIGger GLITch 24-23
spaces and commas 1-5
spelling of headers 1-10
SPOLarity
in TRIG ADV STV 24-64
SPOLL example 4-9
Square Brackets 1-5
SRATe 10-16
*SRE (Service Request Enable) 7-18
SRE (Service Request Enable Register) 4-10
SSAVer 14-19
Standard Event Status Enable Register (SESER) 4-12
Bits 7-6
Default 2-4
Standard Event Status Register
bits 7-8
Standard Event Status Register (ESR) 4-11
Standard Status Data Structure Model 4-2
STARt 11-12
STARt | STOP 21-40
STATistics 20-77
status 1-21
of an operation 4-2
Status Byte
(*STB) 7-20
Status Byte Register 4-8, 4-9
and serial polling 4-9
bits 7-21
Status Registers 1-21, 7-3
Status Reporting 4-2
Bit Definitions 4-3
Data Structures 4-5
Status Reporting Decision Chart 4-19
STATus, in CALibrate command 11-13
*STB (Status Byte) 7-20
STDDev
in MEASure HISTogram command 20-51
STIMe 21-41, 21-43
STOP 8-20
STORe 8-21, 13-10
STORe WAVEform 8-22
storing waveform
sample program 6-15
STRing 14-20
string variables 1-19
example 1-19
string, quoted 14-13
strings, alphanumeric 1-10
STV commands 24-60
SUBTract 16-29
suffix multipliers 1-11, 3-5
suffix units 3-5
summary bits 4-8
SWEep
in TRIGger 24-14
syntax error 29-5
SYSTem
SETup and *LRN 7-11
System Commands 9-2
DATE 9-3
DEBug 9-4
DSP 9-6
ERRor? 9-7
HEADer 9-8
HELP HEADers 9-10
LANGuage 9-12
LONGform 9-13
SETup 9-15
TIME 9-17
System Computer
Returning control to 2-9

**T**

Talker
Code and Capability 2-5
Unaddressing 2-9
TDELta? 19-6
TEDGe
in MEASure command 20-78
temperature and calibration 11-3
TER? (Trigger Event Register) 8-23
termination of message during hardcopy 3-4
Terminator 1-12
Test (*TST) 7-23
TEXT 14-21
THReshold
in MEASure FFT command 20-32

THReshold, and DEFine 20-16, 20-17
TIME 9-17
time and date, setting 9-2
Time Base Commands 23-2
  DELay 23-3
  POSition 23-5
  RANGe 23-6
  REFerence 23-7
  SCALe 23-8
  VIEW 23-9
  WINDow DELay 23-10
  WINDow RANGe 23-13
time base reference
  and DELay 23-3
time buckets
  and POINts? 25-35
time difference between markers 19-6
time information
  of waveform 6-15
time interval, and DELay 23-3
time scale
  operands and functions 16-3
TIMebase POSition
  and DELay 23-4
TITLe? 21-42
TMAX 20-80
TMIN 20-82
TOPBase, and DEFine 20-16, 20-18
transferring waveform data 25-2
  sample program 6-12
transition violation mode 24-109
transmission mode
  and FORMat 25-32
traversal rules 5-7
Tree Traversal
  Examples 5-12
  Rules 5-7
*TRG (Trigger) 7-22
TRG
  bit 7-19, 7-21
  bit in the status byte 4-10
  Event Enable Register 4-4
Trigger
  (*TRG) 7-22
  *TRG status bit 4-4
Trigger Commands 24-2
  TRIG ADV COMM BWID 24-26
  TRIG ADV COMM ENCode 24-27
TRIG ADV COMM LEVel 24-28
TRIG ADV COMM PATTern 24-29
TRIG ADV COMM POLarity 24-30
TRIG ADV COMM SOURce 24-31
TRIG ADV EDLY ARM SLOPe 24-46
TRIG ADV EDLY ARM SOURce 24-45
TRIG ADV EDLY EVENt DELay 24-47
TRIG ADV EDLY EVENt SLOPe 24-49
TRIG ADV EDLY EVENt SOURce 24-48
TRIG ADV EDLY TRIG SLOPe 24-51
TRIG ADV EDLY TRIG SOURce 24-50
TRIG ADV PATT CONDition 24-34
TRIG ADV PATT LOGic 24-35
TRIG ADV STATe CLOCk 24-38
TRIG ADV STATe CONDition 24-39
TRIG ADV STATe LOGic 24-40
TRIG ADV STATe LTYPe 24-41
TRIG ADV STATe SLOPe 24-42
TRIG ADV STV FIELd 24-61
TRIG ADV STV LINE 24-62
TRIG ADV STV SOURce 24-63
TRIG ADV STV SPOLarity 24-64
TRIG ADV TDLY ARM SLOPe 24-55
TRIG ADV TDLY ARM SOURce 24-54
TRIG ADV TDLY DELay 24-56
TRIG ADV TDLY TRIG SLOPe 24-58
TRIG ADV TDLY TRIG SOURce 24-57
TRIG ADV UDTV EDGE 24-68
TRIG ADV UDTV ENUMber 24-69
TRIG ADV UDTV PGTHan 24-70
TRIG ADV UDTV PLTHan 24-71
TRIG ADV UDTV POLarity 24-72
TRIG ADV UDTV SOURce 24-73
TRIG ADV VIOL MODE 24-76
TRIG ADV VIOL PWID DIR 24-81
TRIG ADV VIOL PWID POL 24-80
TRIG ADV VIOL PWID WIDT 24-82
TRIG ADV VIOL PWIDth 24-79
TRIG ADV VIOL SET HOLD CSO 24-94
TRIG ADV VIOL SET HOLD CSO EDGE 24-96
TRIG ADV VIOL SET HOLD CSO LEV 24-95
TRIG ADV VIOL SET HOLD DSO 24-97
TRIG ADV VIOL SET HOLD DSO HTHR 24-98
TRIG ADV VIOL SET HOLD DSO LTHR 24-99
TRIG ADV VIOL SET HOLD TIME 24-100
TRIG ADV VIOL SET MODE 24-86
TRIG ADV VIOL SET SET CSO 24-87
TRIG ADV VIOL SET SET CSO EDGE 24-89
TRIG ADV VIOL SET SET CSO LEV 24-88
TRIG ADV VIOL SET SET DSO 24-90
TRIG ADV VIOL SET SET DSO HTHR 24-91
TRIG ADV VIOL SET SET DSO LTHR 24-92
TRIG ADV VIOL SET SET TIME? 24-93
TRIG ADV VIOL SET SHOL CSO 24-101
TRIG ADV VIOL SET SHOL CSO EDGE 24-103
TRIG ADV VIOL SET SHOL CSO LEV 24-102
TRIG ADV VIOL SET SHOL DSO 24-104
TRIG ADV VIOL SET SHOL DSO HTHR 24-105
TRIG ADV VIOL SET SHOL DSO LTHR 24-106
TRIG ADV VIOL SET SHOL HTIMe 24-108
TRIG ADV VIOL SET SHOL STIMe 24-107
TRIG ADV VIOL TRAN 24-111
TRIG ADV VIOL TRAN SOUR 24-112
TRIG ADV VIOL TRAN SOUR HTHR 24-113
TRIG ADV VIOL TRAN SOUR LTHR 24-114
TRIG ADV VIOL TRAN TYPE 24-115
TRIG EDGE COUPling 24-17

TRIG EDGE SLOPe 24-18
TRIG EDGE SOURce 24-19
TRIG GLITch POLarity 24-22
TRIG GLITch SOURce 24-23
TRIG GLITch WIDTh 24-24
TRIG HOLDoff 24-9
TRIG HTHR 24-10
TRIG HYSTeresis 24-11
TRIG LEVel 24-12
TRIG LTHR 24-13
TRIG SWEep 24-14
TRIGger MODE 24-8
TRIGger EDGE SLOPe 24-15
TRIGger EDGE SOURce 24-15
Trigger Event Register (TRG) 4-10
trigger mode 24-6
ADVanced 24-6
advanced delay 24-43, 24-52
advanced TV 24-59, 24-65
COMM 24-25
delay 24-44, 24-53
EDGE 24-15, 24-16
GLITch 24-20, 24-21
NTSC TV 24-59
PAL-M TV 24-59
pattern 24-33
state 24-37
User Defined TV 24-65
valid commands 24-7
violation types 24-75
triggering
for User Defined TV mode 24-67
truncating numbers 1-11
Truncation Rule 5-5
*TST (Test) 7-23
TSTArt 19-7
TSTOp 19-9
TVOLt 20-83
TYPE query 25-42

U
UDTV commands 24-66
Unaddressing all listeners 2-9
UNITs 12-23, 15-17
in MTESt AMASk command 21-10
units, vertical 12-23, 15-17
UNKnown vertical units 12-23, 15-17
uppercase 1-10
headers 1-10
letters and responses 1-10
URQ bit (User Request) 7-5
User Request (URQ) status bit 4-3
User Request Bit (URQ) 7-5
User-Defined Measurements 20-4
Using the Digitize Command 1-16
USR bit 7-19, 7-21

V
VAMPlitude 20-85
VAVerage 20-87
VBASe 20-89
VDELta? 19-11
version of software, reading 7-9
VERSus 16-30
VERTical 16-31
vertical
axis control 12-2, 15-2
axis offset, and YRANge 26-8
scaling and functions 16-3
scaling, and YRANge 26-9
vertical axis, full-scale 12-21
vertical units 12-23, 15-17
VIEW 8-24, 23-9, 25-44
VIEW and BLANk 8-5
VIOLation MODE 24-76
violation modes for trigger 24-75
VIOLation PWIDth DIRection 24-81
VIOLation PWIDth POLarity 24-80
VIOLation PWIDth SOURce 24-79
VIOLation PWIDth WIDTh 24-82
VIOLation SETup HOLD CSOurce 24-94
VIOLation SETup HOLD CSOurce EDGE 24-96
VIOLation SETup HOLD CSOurce LEVel 24-95
VIOLation SETup HOLD DSOurce 24-97
VIOLation SETup HOLD DSOurce HTHReshold 24-98
VIOLation SETup HOLD DSOurce LTHReshold 24-99
VIOLation SETup HOLD TIME 24-100
VIOLation SETup MODE 24-86
VIOLation SETup SETup CSOurce 24-87
VIOLation SETup SETup CSOurce EDGE 24-89
VIOLation SETup SETup CSOurce LEVel 24-88
VIOLation SETup SETup DSOurce 24-90
VIOLation SETup SETup DSOurce HTHReshold 24-91
VIOLation SETup SETup DSOurce LTHReshold 24-92
VIOLation SETup SETup TIME 24-93
VIOLation SETup SHOLd CSOurce 24-101
VIOLation SETup SHOLd CSOurce EDGE 24-103
VIOLation SETup SHOLd CSOurce LEVel 24-102
VIOLation SETup SHOLd DSOurce 24-104
VIOLation SETup SHOLd DSOurce HTHReshold 24-105
VIOLation SETup SHOLd DSOurce LTHReshold 24-106
VIOLation SETup SHOLd HoldTIMe 24-108
VIOLation SETup SHOLd SetupTIMe 24-107
VIOLation TRANsition 24-111
VIOLation TRANsition SOURce 24-112
VIOLation TRANsition SOURce HTHReshold 24-113
VIOLation TRANsition SOURce LTHReshold 24-114
VIOLation TRANsition TYPE 24-115
VLOWer 20-91
VMAX 20-92
VMIDdle 20-94
VMIN 20-95
voltage at center screen 12-6
voltage information
of waveform 6-15
VOLTS as vertical units 12-23, 15-17
VPP 20-97
VRMS 20-99
VSTArt 19-12
VSTOp 19-14
VTIMe 20-101
VTOP 20-102
VUPPer 20-104

**W**
W, and DATA 25-11
*WAI (Wait-to-Continue) 7-24
Wait-to-Continue (*WAI) 7-24
WATTS as vertical units 12-23, 15-17
waveform
  COUNT and COMPlete? 25-8
  data and preamble 25-3
  SOURce and DATA 25-11
  storing 13-10
  storing time and voltage 6-15
  time and voltage information 6-15
  view parameters 25-45
Waveform Commands 25-2
  BANDpass? 25-5
  BYTeorder 25-6
  CLIPped? 25-7
  COMPlete? 25-8
  COUNt? 25-9
  COUPling? 25-10
  DATA 25-11
  FORMat 25-32
  POINts? 25-35
  PREamble 25-36
  TYPE? 25-42
  VIEW 25-44
  WAVeform SOURce 25-41
  XDISplay? 25-46
  XINCrement? 25-47
  XORigin? 25-48
  XRANge? 25-49
  XREFerence? 25-50
  XUNits? 25-51
  YDISplay? 25-52
  YINCrement? 25-53
  YORigin? 25-54
  YRANge? 25-55
  YREFerence? 25-56
  YUNits? 25-57
waveform memory
  and DATA 25-11
Waveform Memory Commands 26-2
  DISPlay 26-3
  LOAD 26-4
  SAVE 26-5
  XOFFset 26-6
  XRANge 26-7
  YOFFset 26-8
  YRANge 26-9
waveform type
  and COMPlete? 25-8
  and COUNt? 25-9
  and TYPE? 25-42
WAVeforms?
  in MTEST COUNt command 21-20
white space (separator) 1-5
WIDTh
  and GLITch 24-24
  in TRIGger GLITch 24-24
WINDow
  and VIEW 25-44
  DELay 23-10
  in FUNCtion FFT command 16-12
  POSition 23-12
  RANGe 23-13
  SCALe 23-14
WINDow and VIEW 23-9
WORD
  and FORMat 25-33
  Understanding the format 25-28
writing
  quoted strings 14-13
  text to the screen 14-20

**X**
x axis, controlling 23-2
X vs Y 16-30
X1
  in MTESt SCALe command 21-35
X1Position 19-16
X1Position|LLIMit
  in HISTogram WINDow command 18-9
X1Y1source 19-18
X2Position 19-17, 19-22
X2Position|RLIMit
  in HISTogram WINDow command 18-10
X2Y2source 19-19
x-axis
  offset, and XOFFset 26-6
  range, and XRANge 26-7
  units and XUNits 25-51
x-axis duration
  and XRANge? 25-49
XDELta
  in MTESt AMASk command 21-11
  in MTESt SCALE command 21-36
XDELta? 19-20
XDISplay query 25-46
XINCrement query 25-47
XOFFset 26-6
XORigin query 25-48
XRANge 26-7
XRANge query 25-49
XREFerence? 25-50
XUNits query 25-51

**Y**
Y1
  in MTESt SCALe command 21-37
Y1Position 19-21
  in HISTogram WINDow command 18-11
Y2
  in MTESt SCALe command 21-38
Y2Position
  in HISTogram WINDow command 18-12
Y-axis control 12-2, 15-2
YDELta
  in MTESt AMASk command 21-13
YDELta? 19-23
YDISplay? 25-52
YINCrement query 25-53
YOFFset 26-8
YORigin query 25-54
YRANge 26-9
YRANge query 25-55
YREFerence query 25-56
YUNits query 25-57

© Copyright Agilent Technologies 1997-2002. All Rights Reserved.

Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

**Restricted Rights Legend.**

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DOD agencies, and subparagraphs (c) (1) and (c) (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

Agilent Technologies

3000 Hanover Street

Palo Alto, California 94304 U.S.A.

**Document Warranty**

The information contained in this document is subject to change without notice.

**Agilent Technologies makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.**

Agilent Technologies shall not be liable for errors contained herein or for damages in connection with the furnishing, performance, or use of this material.

**Safety**

This apparatus has been designed and tested in accordance with IEC Publication 1010, Safety Requirements for Measuring Apparatus, and has been supplied in a safe condition. This is a Safety Class I instrument (provided with terminal for protective earthing). Before applying power, verify that the correct safety precautions are taken (see the following warnings). In addition, note the external markings on the instrument that are described under "Safety Symbols."

**Warning**

- Before turning on the instrument, you must connect the protective earth terminal of the instrument to the protective conductor of the (mains) power cord. The mains plug shall only be inserted in a socket outlet provided with a protective earth contact. You must not negate the protective action by using an extension cord (power cable) without a protective conductor (grounding). Grounding one conductor of a two-conductor outlet is not sufficient protection.
- Only fuses with the required rated current, voltage, and specified type (normal blow, time delay, etc.) should be used. Do not use repaired fuses or short-circuited fuseholders. To do so could cause a shock of fire hazard.
- Service instructions are for trained service personnel. To avoid dangerous electric shock, do not perform any service unless qualified to do so. Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.
- If you energize this instrument by an auto transformer (for voltage reduction), make sure the common terminal is connected to the earth terminal of the power source.
- Whenever it is likely that the ground protection is impaired, you must make the instrument inoperative and secure it against any unintended operation.
- Do not operate the instrument in the presence of flammable gasses or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.
- Do not install substitute parts or perform any unauthorized modification to the instrument.
- Capacitors inside the instrument may retain a charge even if the instrument is disconnected from its source of supply.
- Use caution when exposing or handling the CRT. Handling or replacing the CRT shall be done only by qualified maintenance personnel.

**Safety Symbols**

Instruction manual symbol: the product is marked with this symbol when it is necessary for you to refer to the instruction manual in order to protect against damage to the product.

Hazardous voltage symbol.

Earth terminal symbol: Used to indicate a circuit common connected to grounded chassis.

**WARNING**

The Warning sign denotes a hazard. It calls attention to a procedure, practice, or the like, which, if not correctly performed or adhered to, could result in personal injury. Do not proceed beyond a Warning sign until the indicated conditions are fully understood and met.

**CAUTION**

The Caution sign denotes a hazard. It calls attention to an operating procedure, practice, or the like, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product. Do not proceed beyond a Caution symbol until the indicated conditions are fully understood or met.

Agilent Technologies
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901

**Product Warranty**

This Agilent Technologies product has a warranty against defects in material and workmanship for a period of three years from date of shipment. During the warranty period, Agilent Technologies will, at its option, either repair or replace products that prove to be defective. For warranty service or repair, this product must be returned to a service facility designated by Agilent Technologies. For products returned to Agilent Technologies for warranty service, the Buyer shall prepay shipping charges to Agilent Technologies and Agilent Technologies shall pay shipping charges to return the product to the Buyer. However, the Buyer shall pay all shipping charges, duties, and taxes for products returned to Agilent Technologies from another country. Agilent Technologies warrants that its software and firmware designated by Agilent Technologies for use with an instrument will execute its programming instructions when properly installed on that instrument. Agilent Technologies does not warrant that the operation of the instrument software, or firmware will be uninterrupted or error free.

**Limitation of Warranty**

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by the Buyer, Buyer- supplied software or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

**No other warranty is expressed or implied. Agilent Technologies specifically disclaims the implied warranties of merchantability or fitness for a particular purpose.**

**Exclusive Remedies**

The remedies provided herein are the buyer's sole and exclusive remedies. Agilent Technologies shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

**Assistance**

Product maintenance agreements and other customer assistance agreements are available for Agilent Technologies products. For any assistance, contact your nearest Agilent Technologies Sales Office.

**Certification**

Agilent Technologies certifies that this product met its published specifications at the time of shipment from the factory. Agilent Technologies further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology, to the extent allowed by the Institute's calibration facility, and to the calibration facilities of other International Standards Organization members.

**About this edition**

This is the third edition of the *Infiniium Oscilloscopes Programmer's Reference.*

Publication number
54810-97076, Mar. 2002

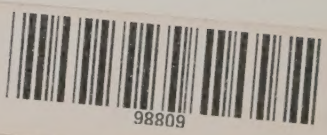
Print history is as follows:
54810-97001, Sept. 1997
54810-97016, Sept. 1998
54810-97031, May 1999
54810-97043, Sept. 1999
54810-97056, January 2000
54810-97059, March 2000
54810-97063, Oct. 2000
54810-97064, Feb. 2001
54810-97076, Mar. 2002

New editions are complete revisions of the manual. Many product updates do not require manual changes; and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual updates.

Agilent Technologies

Agilent Technologies
Printed in the USA

PROGRAMMER'S GUIDE-ENGLISH

54810-99011

Assembled in U.S.A